Tomas Bata University in Zlín
Faculty of Applied Informatics

Ing. Zuzana Oplatková

Doctoral Thesis

Metaevolution – Synthesis of Evolutionary Algorithms by means of Symbolic Regression

Study-branch: Technical Cybernetics
Supervisor: assoc. prof. Ivan Zelinka

Zlín, Czech Republic, 2007

Acknowledgements

I would like to express my warm thanks to:

- my supervisor, ***assoc. prof. Ivan Zelinka***, for his support and invaluable discussions during my study and research on this thesis,
- ***MSc. Donald Davendra*** who revised the English in this dissertation,
- ***Radek Bernátík*** for his love which helps me a lot,
- ***colleagues and friends***, mainly ***František Můčka*** who disscussed with me for long hours on different topics concerned to this thesis,
- and my ***parents*** who supported me during my studies

...dedicated to hurried time...
...to catch the eyewink in our lives as the global extreme of fast world...

RESUMÉ

Toto pojednání je zaměřeno na vysvětlení, jak může být Analytické programování použito pro vytvoření nových optimalizačních algoritmů, pravděpodobně evolučního charakteru. Evoluční algoritmy jsou nástrojem pro optimalizaci složitých úloh. Jedním z cílů práce je ukázat, že je možné syntetizovat účinný algoritmus, který bude založený na evolučních principech. Toto všechno se skrývá pod pojmem metaevoluce, jak ji chápeme z našeho pohledu. Metaevoluce podle předchozích přístupů znamená hledání nejlepších algoritmů, jejich operátorů a jejich nastavení pro daný problém. Prakticky to znamená, že nějaký algoritmus ladí jiný algoritmus pro nejlepší chování na daném problému. Náš přístup je jiný, nehledáme pouze nastavení algoritmu, ale používáme metaevoluci k vytvoření kompletně nového algoritmu.

Nejprve jsou popsány současné metody pro regresi – Genetické programování, Gramatická evoluce a Analytické programování. Poslední z nich je použito pro simulace v této studii. V další sekci jsou popsány evoluční algoritmy, které byly použity pro simulace a také k porovnání jejich robustnosti.

Následující část popisuje simulační experimenty, které byly provedeny Analytickým programováním. Nejprve jsou zachyceny simulace pro fitování dat, což znamená použití regrese k nalezení vhodného matematického zápisu. Tato formule, která je vystavěná z jednoduchých funkcí jako je plus, minus nebo proměnné x či konstanty, by měly proložit data co nejlépe. Simulace potvrdily, že AP je schopné pracovat s tímto typem problémů, dokonce s menším počtem ohodnocení účelové funkce než GP. Pro ukázku, že fitování dat funguje, byly provedené 4 studie - Quintic, Sextic, Three Sine and Four Sine problém.

Druhou úlohou bylo navržení elektronických obvodů. Cílem bylo najít zapojení obvodu funkcí odpovídající dané pravdivostní tabulce. Z jednoduchých funkcí jako je And, Nand, Or a vstupů byly vytvořeny konečné výrazy k – symetrického a k – sudého problému. Hodnoty k byly postupně nastaveny na hodnoty 3 až 6 pro oba problémy.

Poslední úlohou, která měla prokázat, že AP je schopné pracovat i s lingvistickými výrazy jako jsou např. příkazy pro robota, bylo nastavení optimální trajektorie pro robota. Na definované cestě bylo rozmístěno jídlo a

také překážky. Umělý mravenec, který byl navržený v původní úloze Kozou, měl sníst všechno jídlo na této trase. Mravenec musel také překonat umístěné překážky, které v tomto případě byly políčka na cestě, které neobsahovaly jídlo.

Tyto předpoklady vedly k hypotéze, že AP je schopno vytvořit i nový optimalizační algoritmus, pravděpodobně evolučního charakteru. Sekce 6 popisuje vývoj od první studie tohoto druhu až po simulace s více operátory a ve vyšších dimenzích. Na začátku jsme si vybrali Diferenciální evoluci (DE), ze které jsme separovali její operátory na samostatně pracující moduly. Tyto operátory byly nastaveny jako jednoduché základní funkce pro AP. Během simulace byly vyšlechtěna jak úspěšná včetně originální DE, tak i neúspěšná řešení.

Následující krok bylo použití více operatorů z dalších evolučních a stochastických algoritmů jako je SamoOrganizujícíc se Migrační Algoritmus (SOMA), Horolezecký algoritmus a Simulované žíhání. V tomto případě jsme také použili vylepšenou verzi účelové funkce. S ohledem na řád jednotlivých hodnot z testovacích funkcí jsme změnili výpočet hodnoty na rozdíl mezi nalezeným a globálním extrémem. To také umožnilo snadnější penalizaci týkajícíc se počtu ohodnocení účelové funkce.

Simulace v této sekci byly provedeny v 2 dimensionálním prostoru. To vedlo k třetímu kroku a to použití benchmark funkcí ve vyšších dimenzích jako kritérium v AP. Obdržené výsledky z vícerozměrových testovacích funkcí - 4 nalezené algoritmy – byly aplikovány na 16 testovacích funkcí ve 2, 20 a 100 dimenzionálním prostoru. Celkem bylo provedeno 192 simulací, z nichž každá byla 100krát zopakována. Výsledky jsou presentovány formou tabulek a grafů v příloze.

Z výsledků lze usoudit, že nalezené algoritmy jsou schopné optimalizovat multimodální funkce. Není možno říci, který z nich zvítězil, jednak kvůli ne zcela totožný počtům ohodnocení účelové funkce, ale také proto, že žadný algoritmus nevyhrál ve všech testovacích funkcích. Soutěžili dokonce i v různých dimenzích v rámci jedné funkce.

Budoucí výzkum je otevřený v oblasti přidávání operátorů, ladění parametrů nalezených algoritmů nebo syntéza nového evolučního operátoru.

Klíčová slova: symbolická regrese, Analytické Programování (AP), evoluční algoritmy, nové evoluční algoritmy

ABSTRACT

This thesis is aimed at the explanation as to how Analytic Programming could be used for the creation of new optimizing algorithms, probably of evolutionary character. Evolutionary algorithms are tools for the optimization of difficult tasks. The principle of this thesis is to show that it might be possible to synthesize a powerful algorithm based on evolutionary ideas. The name of this thesis – metaevolution – covers all these ideas. Metaevolution is, according to previous approaches, determining the optimal evolutionary algorithm, best types of evolutionary operator and their parameter setting for a given problem. It means basically, that one evolutionary algorithm tunes another one. But this approach is novel. We use metaevolution for creating a new algorithm completely, not only for setting of its parameters.

Firstly, tools for regression are described – Genetic Programming, Grammatical Evolution and Analytic Programming (AP); the last one is used in this study. Other tools, which are seen here, depict evolutionary algorithms which were used for simulations purposes in order to also to compare their robustness.

The following part describes projects which were conducted by Analytic Programming. Firstly, simulations of fitting measured data are mentioned, which implies the use of regression to finding a suitable mathematical formula. This complex formula, based on simple functions like plus, minus or variables “x” and constants, should fit the data as closely as possible. The simulations proved that AP is able to perform such kinds of computations even in a smaller number of cost function evaluations compared to GP (four problems were carried out – Quintic, Sextic, Three Sine and Four Sine problem - to show that this type of regression works).

The second task was to design electronic circuits. The aim was to find a configuration of circuits according to the truth table. Whole expression of k-symmetry and even-k-parity problems were created from simple functions like And, Nand, Or and inputs. Values 3 to 6 for both types of problems were set up for k.

The last task, which proved that AP is able to work also with linguistic terms like commands for robot, was setting of optimal trajectory for robot. In the defined problem trail pieces of food were placed including some obstacles.

The artificial ant, originally defined by Koza, should eat all the food on such a trail while overcoming all obstacles.

These presumptions led to the hypothesis that a new algorithm of evolutionary character can be created by Analytic Programming. Section 6 describes the progress from the first study of the creation of a new evolutionary algorithm to the simulations with more operators and higher dimensional systems. At the onset Differential Evolution (DE) was taken and its operators were separated into modules which are able to work independently. These operators were set up as simple functions for successful evaluations of AP. During the run non successful solutions as well as the original Differential Evolution and other successful solutions were found.

The next step continued with more operators from other evolutionary and stochastic algorithms such as Self-Organizing Migrating Algorithm, Hill Climbing and Simulated Annealing. In this case also a new design of cost function was. With respect to the order of obtained cost values, the measurement was changed to minimize the difference between found extreme and the global one. This also affords to apply penalization concerned to cost function evaluations. Simulations in this section were performed in 2 dimensional space. This led to the third step, to use high dimensional benchmark functions as criterion in AP. The obtained results from higher dimensional test functions were then applied on 16 benchmark function in 2, 20 and 100 dimensional space for 4 found algorithms. Altogether 192 simulations were carried out in 100 times repetition, it means nearly 4 milliards of cost function evaluations. Results are depicted in tables and graphs in the Appendix.

From results obtained, it can be stated that found algorithms are able to optimize multimodal functions. However, it is not possible to say which one was better because each won in some cases. They compete even inside one benchmark function in different dimensions.

Future research is open to add more operators, to tune parameters of found algorithms or to try to synthesize a new evolutionary operator itself.

Keywords: Symbolic Regression, Analytic Programming, evolutionary algorithms, new evolutionary algorithms

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS AND ABBREVIATIONS

$x_{i,j}^{ML}$	j – parameter of i – individual in migration loop ML
$x_{L,j}^{ML}$	j – parameter of Leader in migration loop ML
$x_{i,j,START}^{ML}$	Start position of j – parameter of i – individual in migration loop ML
AllToAll	Strategy of SOMA, all individual search in the direction to all individuals
AllToAllAdaptive	Strategy of SOMA, all individuals search in the direction to all by means of adaptive way
AllToOne	Strategy of SOMA, all individuals search in the direction to Leader
AllToOneRand	Strategy of SOMA, all individuals search in the direction to one randomly selected
AP	Analytic Programming
CF	Cost Function
CFE	Cost Function Evaluations
CompleteHC	Operator of HillClimbing algorithm
CompleteSA	Operator of Simulated Annealing
CR	Control parameter of DE, crossover constant

CrossDEBin	Crossover operator of Differential Evolution – Bin version
CrossDEExp	Crossover operator of Differential Evolution – Exp version
CrossoverDE	Crossover operator of Differential Evolution – DERand1Bin, previous name
CV	Cost value
DE	Differential Evolution
Dim	Dimensionality of given problem (number of arguments)
EA	Evolution Algorithms
F	Control parameter of DE, mutable constant
F(ind)	Fitness value in GA
f_{max}	Maximum value of cost function
f_{min}	Minimum value of cost function
GA	Genetic Algorithms
GE	Grammatical Evolution
Generations	Stopping parameter of DE and GA, number of loops in all evolution
GFS	General Functional Space
GFS_{0arg}	Functions with 0 arguments in GFS, i.e. constants and variables

GFS_{1arg}	Functions with 1 argument in GFS (e.g. Sin, Cos, Tan..)
GFS_{2arg}	Functions with 2 arguments in GFS (e.g. +,-,/….)
GFS_{3arg}	Functions with 3 arguments in GFS
GFS_{All}	Set of all functions in general functional space
GP	Genetic Programming
HC	Hill Climbing
Leader	Individual with minimal cost value in migration loop in SOMA
Migrations	Stopping parameter of SOMA, number of migration loops
MinDiv	Stopping parameter of SOMA, minimal accepted error between the best and worst individual in population
MutateDEBest1	Mutation operator of Differential Evolution – DE Best1Bin
MutateDEBest2	Mutation operator of Differential Evolution – DE Best2Bin
MutateDECurrentToBest	Mutation operator of Differential Evolution – DECurrentToBestBin
MutateDERand1	Mutation operator of Differential Evolution – DERand1Bin
MutateDERand2	Mutation operator of Differential

	Evolution – DE Rand2Bin
MutationDE	Mutation operator of Differential Evolution – DERand1Bin, previous name
NP, NPIIII	Number of individuals in population
PathLength	Control parameter of SOMA; it determines the stopping position of the movement of an individual
PopSize, PopSizeIIII	Number of individuals in population
Population	Matrix NP x number of arguments of an individual
PRT, PRTIIII	Control parameter of SOMA; perturbed vector PRTVector is generated according to it. It has an interaction with the movement of an individual.
$PRTVector_j$	Vector of zeros and ones, it interacts with the movement of an individual
SA	Simulated Annealing
SelectDE	Selection operator of Differential Evolution
SelectHC	Selection operator of HillClimbing – random point in the Cost Function
SelectionDE	Selection operator of Differential Evolution, previous name
SelectLeaderSOMA,	Selection operator of SOMA – the best individual in the population

SelectSOMARandLeader	Selection operator of SOMA – random individual in the population
SOMA	Self-Organizing Migrating Algorithm
SOMAATORandWithoutPRT	SOMAATORand version operator, perturbation not included
SOMAATORandWithPRT	SOMAATORand version operator included perturbation
SOMAATOWithoutPRT	SOMAATO version operator, perturbation not included
SOMAATOWithPRT	SOMAATO version operator included perturbation
Specimen	Individual with definitions of parameters (type – integer, real.., interval)
Step	Control parameter of SOMA, length of step of an individual during search
t	Parameter in SOMA – through steps

1. INTRODUCTION AND STATE OF ART

Optimization is one of these words which is used almost every day in different fields of human activities. Everybody wants to maximize profit and minimize cost. This means optimizing in every task of industry, transportation, medicine, everywhere. For these purposes, we need to have suitable tools which are able to solve very difficult and complicated problems. As previous years proved, use of artificial intelligence and soft computing contribute to improvements in a lot of activities. One of such tools of soft computing are evolutionary algorithms [1].

Evolutionary algorithms are a group of algorithms which use their special operators as mutation, crossover and others to find an ideal solution. Possible candidates are defined by a cost function which arguments are values of each solution. The best one is in the global extreme – maximum or minimum [1], [2].

These evolutionary algorithms have been known for decades and live through the advancement from the weaker ones to more robust ones which are used with success in a lot of tasks nowadays. Since their first appearance there is quite long queue of representatives: Genetic Algorithms [3], Differential Evolution [4], Self-Organizing Migrating Algorithm [5], Particle Swarm Intelligence [6], Ant Colony Optimization [7], Artificial Immune system [8]. In optimization, algorithms belongs also to some stochastic and deterministic ones: Hill Climbing [9], Simulated Annealing [10], Monte Carlo [2] and a lot of others or their mutations [11].

These techniques promise fast optimization compared to classical mathematical approach. On the other hand, also between these optimization techniques is possible to find better and worse. Their behaviour were described in a lot of references. And the research in this area is still full of white places. There is wide field of possible applications as tuning of parameters, making of comparisons, trying to find new ones somehow.

There exist special tools which are connected with evolutionary algorithms and are able to work with symbolic regression. Nowadays, mainly three are known for that – Genetic Programming [12] - [14], Grammatical Evolution [15] - [17] and superstructure of evolutionary algorithms – Analytic

Programming [18] - [26]. These techniques can produce a complex formula from basic functions according to required behaviour of function in the case of mathematical data set, of an electronic circuit, trajectory of robots, etc.

Also, some other approaches to the field of symbolic regression can be found – either based only on evolutionary techniques or hybrid ones. Interesting investigations using symbolic regression were showed by Johnson [27] working on Artificial Immune Systems and Salustowicz in Probabilistic Incremental Program Evolution (PIPE) [28] which generates programs from an adaptive probability distribution over all possible programs. To Grammatical Evolution foreruns GADS which solves the approach to grammar [29], [30]. Also from evolutionary algorithm artificial immune systems evolved the artificial immune system programming for symbolic regression [31]. Approaches which differ in representation and grammar are described in gene expression programming [32], multiexpression programming [33], meta-modelling by symbolic regression and pareto Simulated Annealing [34]. To the group of hybrid approaches, belongs mainly numerical methods connected with evolutionary systems, e.g. [35].

Then the idea to connect evolutionary algorithms with techniques for symbolic regression came up. The aim is to try to create new evolutionary algorithms which will be very robust and will be used for difficult tasks faster and with higher quality than current algorithms are able to do at present.

This work is divided into five main numbered chapters.

The first chapter gives overview in the research area of symbolic regression by means of tools of artificial intelligence and evolutionary algorithms whereas *the second chapter* formulates the main goals of this dissertation.

The section *number three* is focused on the theoretical knowledge about the symbolic regression and its tools and similarly in section *four* description of evolutionary algorithms used in the work can be found.

The fifth part of the work shows simulation results which had forerun the simulations themselves connected to the main topic of the dissertation – synthesis evolutionary algorithms by means of symbolic regression.

The sixth part offers the progress from the beginning to the final results of the creating evolutionary algorithms, the discussion of the obtained results and conclusion of the achieved goals in this work.

In the *final part* can be found the discussions and conclusions of achieved goals of dissertation and view to the future field in this research.

Tables, figures and equations are numbered recursively within a chapter and literature is referred to in square brackets.

2. DISSERTATION GOALS

The aim of the work is to apply and verify that it is possible to create new evolutionary algorithms by means of symbolic regression with a tool of artificial intelligence – Analytic Programming (AP). Preparation steps were done with several types of tasks chosen from literature to find out the performance of AP. After that we were able to run simulations to find new evolutionary algorithms.

The steps leading to this dissertation could be summarized as follows:

- to prove that Analytic Programming is able to do symbolic regression,
- to prove that Analytic Programming is also able to work with linguistic terms not only with numeric values or mathematical operators,
- to try to create a new optimization algorithm, probably of evolutionary character, possibly with AP,
- to define several operators of evolutionary algorithms (like crossover, mutation, perturbation from SOMA, and others) which will be used as simple functions for AP,
- to define restrictions in Cost Function such as the inclusion of the number of cost function evaluation into quality of solution,
- to try to create an evolutionary algorithm which will be at least as robust as some current algorithms are and further to compare its behaviour with current ones,
- to give comparisons between created and current evolutionary algorithms

THEORETICAL FRAMEWORK

3. SYMBOLIC REGRESSION

In statistics, regression is a method of curve fitting, i.e. finding a curve which matches a series of data points and possibly also other constraints. It is done by means of regression analysis. Two types of regression are used – linear and nonlinear, which depend on data sets. The final formula, which fits data as close as possible, is done using classical mathematical and statistical techniques [36].

Symbolic regression in the context we use, implies to create a final formula from basic simple functions. This procedure can be used for mathematical and also for non mathematical fields.

This approach was firstly introduced by John Koza in Genetic Programming [12] - [14], then in Grammatical Evolution [15] - [17] by Conor Ryan and also by Ivan Zelinka in Analytic Programming [18] - [25].

3.1. Genetic Programming

Genetic programming was introduced at the end of the 1980's by John Koza [12] - [14]. He suggested modification of genetic algorithm and he named it Genetic Programming. In this concept a new population is bred not in the normal numerical way but in an analytical way. It means that the solution of such breeding is not values of parameters but a function itself.

According to genetic algorithms, each value is called gene, similarly to nature. Genes in GP are not represented by integers or real values but parameters in chromosome string are functions themselves. In the simplest version there are variables, constants, basic arithmetical functions and elementary functions. From this group a function, e.g. x*(1+x) can be made. This can be sought in a parse tree as seen in Fig. 3.1, where the top is called the root of the tree.

Interpreting of the parse tree is easy. During the run the function x * (1 + x) is evaluated through this tree from the bottom to the top.

In GP, operators of crossover and mutation are used as in genetic algorithms [37] - [39]. But here, whole parts of a tree are changed in the case of mutation (Fig. 3.2) or crossed (Fig. 3.3).

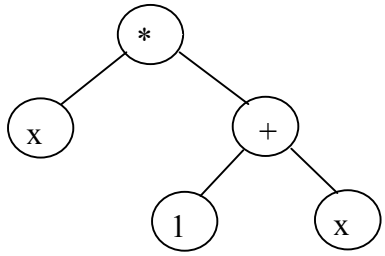


Fig. 3.1: Parse tree

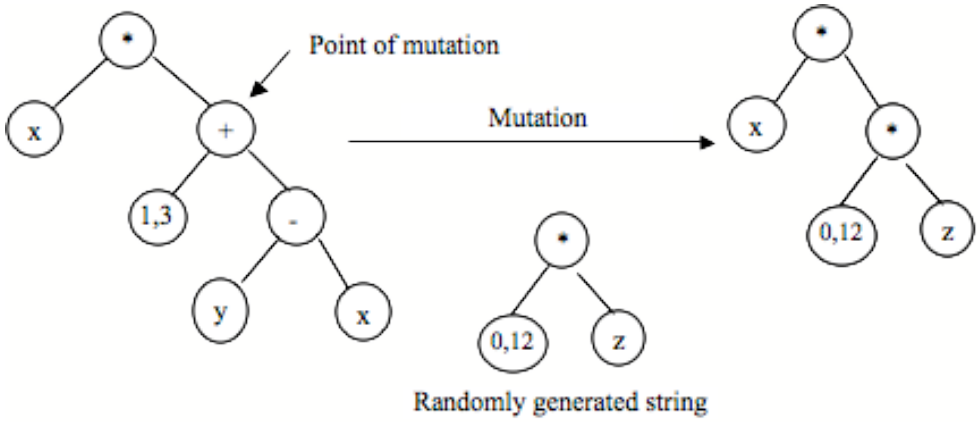


Fig. 3.2: Mutation in Genetic Programming

Another approach to GP is enforcing dimensional constraints through formal grammar. It restricts GP search space to dimensionally admissible laws [40]. Another investigation which adjusts GP to achieve improved predictive performance and reliability of the induced expressions was presented in [41], [42].

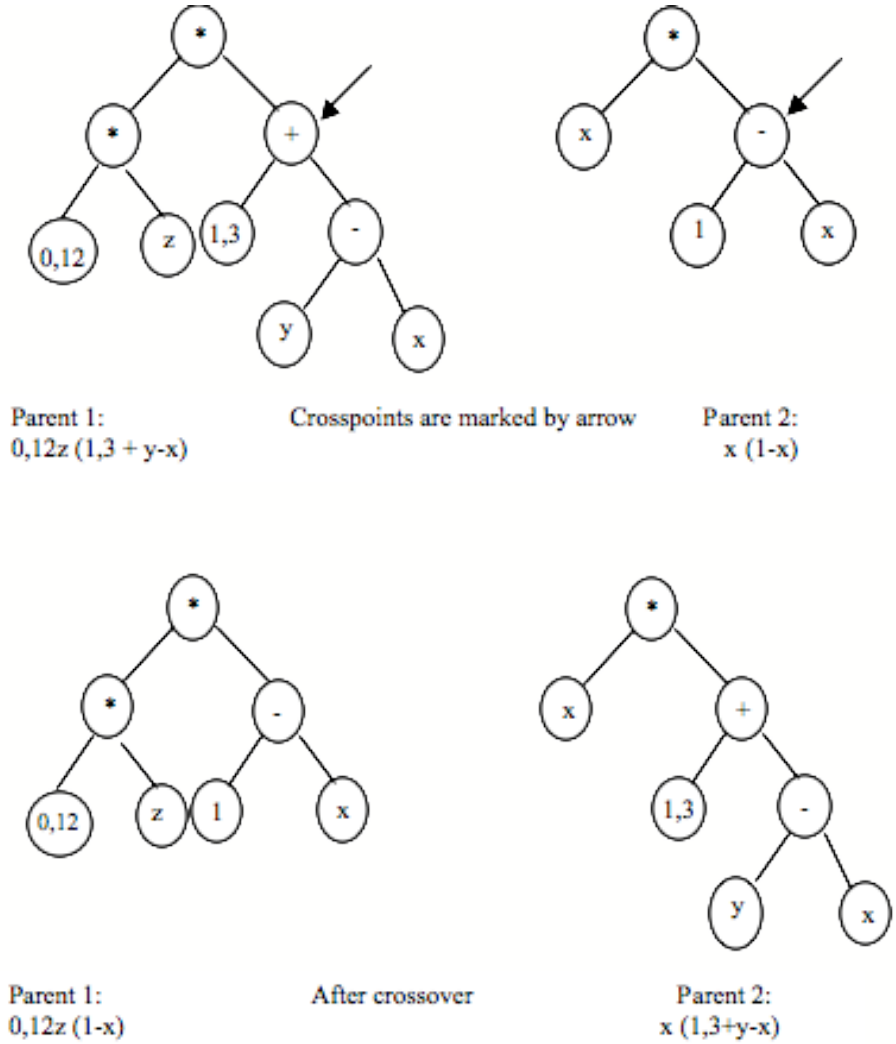


Fig. 3.3: Crossover in Genetic Programming

3.2. Grammatical Evolution

Grammatical evolution (GE) is another tool for doing symbolic regression by means of computers. The advantage of this tool, compared to GP, is that GE can evolve complete programs in an arbitrary programming language [15]

- [17] using a variable – length binary string. It uses Backus Naur Form grammar definition for mapping process to a program. GE performs the whole process on a variable – length binary strings. A mapping process is employed to generate programs in any language by using the binary strings to select production rules in a Backus Naur Form (BNF) grammar definition. The result is the construction of a syntactically correct program from a binary string that can then be evaluated by a fitness function [15].

Variable-length binary string genomes are used with each codon representing an integer value, where codons are consecutive groups of 8 bits in order to make the genetic code degenerate. The integer values are used in a mapping function to select an appropriate production rule from the BNF definition. The numbers generated always represent one of the rules that can be used at that time.

Below is an example of BNF definition, where N is a set of nonterminals and T is set of terminals.

N ={expr, op, pre_op, var}
T = {Sin, + , - , / , * , X , 1.0}}

and can be represented as:

A) <expr> : : = <expr> <op> <expr> (0)
| (<expr> <op> <expr>) (1)
| <pre-op> (<expr>) (2)
| <var> (3)

B) <op> : : = + (0)
| - (1)
| / (2)
| * (3)

C) <pre-op> : : = Sin

D) <var> : : = X (0)
| 1.0 (1)

In Table 3.1, numbers of possibilities for each rule are given. The mapping starts with reading codons of 8 bits [15] to generate a corresponding integer value, from which an appropriate production rule is selected by using the mapping function (3.1).

Table 3.1: The number of choices available from each production rule

Rule type	**Choices**
A	4
B	4
C	1
D	2

$$\text{Rule} = (\text{Codon integer value}) \ \text{MOD} \ (\text{Number of choices for the current non-terminal}) \qquad (3.1)$$

Fig. 3.4 shows an example of the individual with content of integer values which were generated from 8 bit binary number (codon).

220	40	16	203	101	53	202	203	102	55	220	202	19	130	37	202	203	32	39	202	203	102

Fig. 3.4: An example of an individual for GE

The first codon is 220. If we apply eq. (3.1) we obtain value 0. That means we use rule A with its terminal 0. It represents an inscription A.0. Our program looks like

<expr><op><expr>

Then we continue with the left-most non-terminal which is <expr>. We take the second codon from the individual and apply the formula (3.1) again,

i.e. 40 MOD 4. We obtain 0. <expr> is replaced by <expr><op><expr>. The result is following

<expr><op><expr><op><expr>

The next step is again at the <expr>. For the third time we obtain by reading codon the rule A.0.

<expr><op><expr><op><expr><op><expr>

Now the left-most <expr> is determined by codon with value 203 which gives after formula (3.1) rule A.3 which is <var>.

<var><op><expr><op><expr><op><expr>

The next codon will then determine the value of var; there are 2 possibilities. 101 MOD 2 gives then rule D.1 which has value 1.0. Our expression then results in

1.0 <op><expr><op><expr><op><expr>

Next codon will then determine what <op> will become. We have 53 MOD 4 which is 1 which stands for minus. The next <expr> has to be expanded by the codon 202 that is 202 MOD 4 = 2. We get following

1.0 - <pre-op>(<expr>)<op><expr><op><expr>

Because <pre-op> has only one possibility we obtain

1.0 – Sin (<expr>)<op><expr><op><expr>

Then we can continue similarly as before until we end with this final formula.

*1.0 – Sin(x)*Sin(x) - Sin(x)*Sin(x)*

The program is finished when all non-terminals are replaced by terminals. If codons are run out earlier then they are used cyclically from the beginning [15]. The above description is for mapping from codons to final formula in GE. During evolutionary process mutation and crossover operators are used like in genetic algorithms.

3.3. Analytic Programming

3.3.1. Description

Basic principles of the AP were developed in 2001 [5]. Until that time only GP and GE and their mutations existed. GP uses Genetic Algorithms while AP can be used with any evolutionary algorithm, independently on individual representation. To avoid any confusion based on the use of names according to the used algorithm, name - Analytic Programming was chosen by the author, because AP stands for synthesis of analytical solution by means of evolutionary algorithms [18] - [25].

According to the authors of AP, it was inspired by numerical methods in Hilbert spaces (space with mutually orthogonal functions) [43] and by GP. Principles of AP [23] are somewhere between these two philosophies. From GP an idea of evolutionary creation of symbolic solutions is taken into AP while from Hilbert spaces an idea of functional spaces and building of resulting function by means of a searching process usually done by numerical methods like Ritz or Galerkin is adopted. AP as well as GP is based on a set of functions, operators and so-called terminals, which are usually constants or independent variables like for example:

- functions: Sin, Tan, Tanh, And, Or
- operators: +, -, *, /, dt,…
- terminals: 2.73, 3.14, t,…

All these “mathematical” objects create a set which AP tries to synthesize into an appropriate solution form. Main principle (core) of AP is based on

discrete set handling [5], originally proposed in [44], see Fig. 3.5 and Fig. 3.6.

Discrete set handling shows itself as universal interface between EA and a symbolically solved problem. This is why AP can be used almost by any evolutionary algorithm.

Briefly said, in AP, individuals consist of non-numerical expressions (operators, functions,…) as described above, which are in evolutionary process represented by their integer indexes (Fig. 3.5). The index then serves like a pointer into the set of expressions and AP uses it to synthesize resulting function-program for cost function evaluation [22].

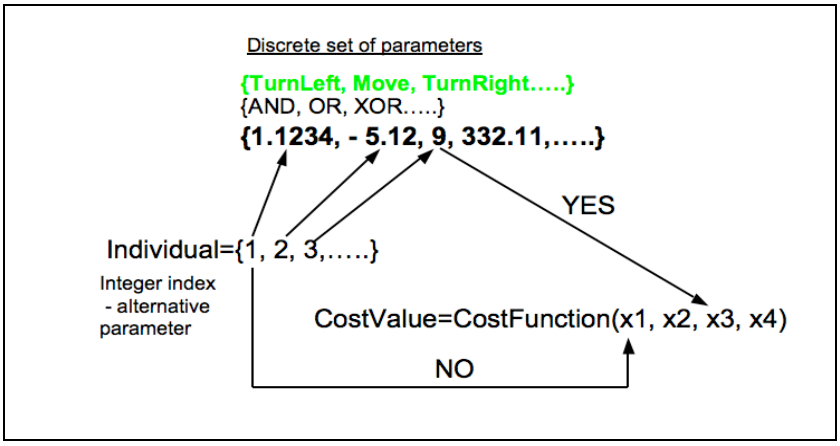


Fig. 3.5: Discrete set handling

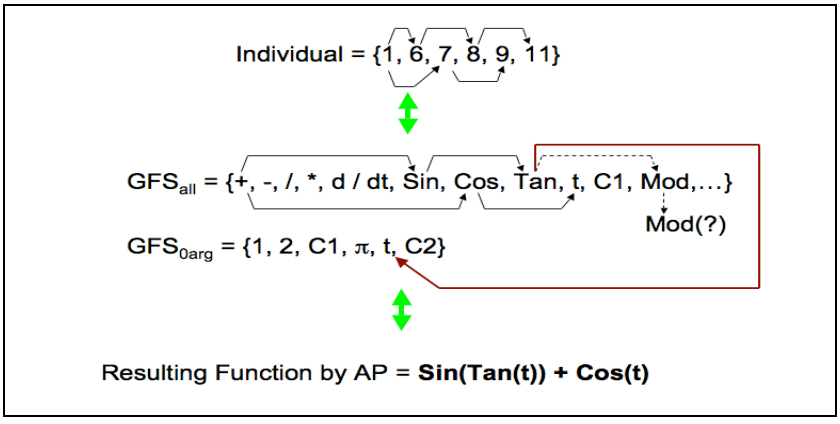


Fig. 3.6: Main principles of AP

The functionality of the discrete set handling can be on the concrete example in Fig. 3.6 described as follows.

All simple functions and operators are in so called General Function Set (GFS) [22]. Each simple function, operator or variable is then divided into groups according to the number of arguments which can be inserted during the evolutionary process to subset GFS_{3arg}, GFS_{2arg}...GFS_{0arg} (Fig. 3.7).

GFS_{all} = {+, -, /, *, Sin, Cos, Tan, t, C1,1, π, BetaRegularized, Log, Mod,…}

GFS_{3arg} = {BetaRegularized, …..}
GFS_{2arg} = {+, -, /, *, Log, Mod,…..}
GFS_{1arg} = {Sin, Cos, Tan, …..}
GFS_{0arg} = {1, C1, π, t,…}

Fig. 3.7: Example of set of General Functional Set and its subsets

The individual consists of 6 arguments (indices, pointers to GFS). The first index is 1, meaning that + is taken from the set of functions GFS_{all}. Function plus has two arguments therefore indexes 6 and 7 are arguments of plus (expression (3.2).

$$6 + 7 \tag{3.2}$$

Index 6 is then replaced by Sin and index 7 by Cos (expression (3.3)).

$$Sin + Cos \tag{3.3}$$

Sin and Cos are one-argument functions. After index 7 follows index 8, which is replaced by Tan (expression (3.4)). Tan is also one-argument function.

$$Sin(Tan) + Cos \tag{3.4}$$

After index 8 the individual takes index 9, which is replaced by “t” and this is an argument of Cos (see expression (3.5)).

$$Sin(Tan) + Cos(t) \tag{3.5}$$

If the last index is 9, the process would finish easily. Under this index would be a variable "t", and this is an argument of Tan. The resulting function mapping by AP would be then expression (3.6).

$$Sin(Tan(t)) + Cos(t) \tag{3.6}$$

But in our case there is function Mod. This needs an argument to work properly. AP will not allow this. Instead of Mod the AP will jump into the subspace in this case directly to GFS_{0arg}. In other cases it is counted as if it can be used also as an operator with more arguments. In the GFS_{0arg} there was found 11^{th} element which is "t". So we will obtain again the expression (3.6).

This description was shown on mathematical operators and objects as functions, variables etc. for simplicity. But it can be used as linguistic terms which must then be suitably transformed in the cost function to the numerical value because of evolutionary algorithms. The usage of algorithms to find a final formula is necessary as mentioned in the introduction. They need numerical value as the measurement of quality of the solution.

Analytic programming was used for e.g. in solving following problems:

- sextic, quintic, 3sine, 4sine problem [20] with the use of algorithms of Simulated Annealing (SA) [10], Genetic Algorithms (GA) [3], Differential Evolution (DE) [44], [4] and Self-Organizing Migrating Algorithm (SOMA) [5]
- Boolean symmetry and parity problems [21], [22], again with SA, GA, DE and SOMA
- Solving of ordinary differential equations (ODE): u''(t) = cos(t), u(0) = 1, u(π) = -1, u'(0) = 0, u'(π) = 0 [43], 100 times repeated, in that case AP was looking for suitable function, which would solve this case of ODE, by DE and SOMA in [18], [19]
- Solving of ODE: ((4 + x)u''(x))'' + 600u(x) = 5000(x-x2), u(0)=0, u(1)=0, u''(0)=0, u''(1)=0, Again as in the previous case, AP was used to synthesize a suitable function – solution of this kind of ODE. This ODE was used from and represents a civil engineering problem in reality, [18]

- Setting an optimal robot trajectory [23], [24] with algorithms SOMA, DE and SA
- Synthesis of neural networks [25] with algorithm SOMA
- Chaos synthesis [26]

3.3.2. Versions of AP

Today, AP exists in three versions. In all three versions the same sets of functions, terminals, etc. as Koza use in GP [12] - [14] are necessary for the program synthesis. The second version (AP_{meta}, lets call the first version AP_{basic}) is modified in the sense of constant estimation. For example, when Koza uses in so called sextic problem [12], randomly generated constants, AP uses only one, called "K", which is inserted into the formula at various places by evolutionary processing. The function can look like as in (3.7).

$$\frac{K}{x + Kx} \tag{3.7}$$

When the program is synthesized, then all "K" are indexed so that K_1, K_2, …, K_n, are obtained in formula (3.8), and then all K_n are estimated by second evolutionary algorithm.

$$\frac{K_1}{x + K_2 x} \tag{3.8}$$

Because EA "works under" EA (i.e. EA_{master}►program►K indexing ►EA_{slave}►estimation of K_n), this version is called AP with metaevolution - AP_{meta}. As this version was quite time consuming, another modification of AP_{meta} was done extending the second version by estimation of K. It is done by suitable methods of nonlinear fitting from the environment Mathematica (www.wolframresearch.com) (AP_{nf}). This method has shown the most promising performance when unknown constants are present, so some comparative simulations were performed using third version - AP_{nf} in article [22].

3.4. Comparison

The above described tools have some elements similar and some that are unique. To the same elements belongs the aim of the creation of the complex formula, which fits data or required behaviour as well as possible. Tools used evolutionary algorithms to their successful run. On the other hand, there are also some disadvantages within each tool.

The first thing, which is different for these tools, is the use of evolutionary algorithms. GP can use only genetic algorithms; it is basically the same as genetic algorithm because the simple functions are directly set up inside the individuals. Then the operators of GA are applied. GE is different because it was not used with GAs only, but also with a few strategies with binary representation of individuals. The last tool, AP, is able to use arbitrary evolutionary algorithm because of its structure and techniques of manipulating with arguments in individuals – discrete set handling as described above.

The programming language is related to this topic as well. GP was used in LISP programming language which enables easy manipulation with symbolically written function inside genetic algorithm. The other two tools are able to be implemented in any programming language.

The fact, that AP can be implemented in arbitrary programming language and it can use arbitrary evolutionary algorithms (e.g. DE, SOMA, GA, SA) was the most important reason for the choice of AP for other experiments. The quality of results depends on the powerful evolutionary algorithm performance. The robustness of the method depends on the choice of evolutionary algorithm. From the experiments, we accomplished, it was found that Self-Organizing Migrating Algorithm [5] and Differential Evolution [44], [4] are very powerful algorithms which will be described in the next section. Also to show that AP is able to cooperate with other evolutionary algorithms in this work simulations with Simulated Annealing [2], [10] and Genetic Algorithms [37] - [39] will be also carried out.

3.5. Other possible approaches

Other interesting investigations using symbolic regression were showed by Johnson [27] working on Artificial Immune Systems and Salustowicz in

Probabilistic Incremental Program Evolution (PIPE) [28] which generates programs from an adaptive probability distribution over all possible programs.

To Grammatical Evolution foreruns GADS which solves the approach to grammar [29], [30]. Also from evolutionary algorithms artificial immune systems came up the artificial immune system programming for symbolic regression [31].

There are three other approaches – gene expression programming [32], multiexpression programming [33], meta-modelling by symbolic regression and pareto Simulated Annealing [34] and also hybrid methods which uses numerical methods with evolutionary approach [35].

4. OPTIMIZATION ALGORITHMS - EVOLUTIONARY ALGORITHMS

Optimization algorithms – mainly evolutionary algorithms are a necessary part of the above described tools and can be used independently. Here, an overview only of algorithms, which were used in further simulations, will be given.

Division of optimization algorithms might be as follows. This is not the only one point of view on that [1].

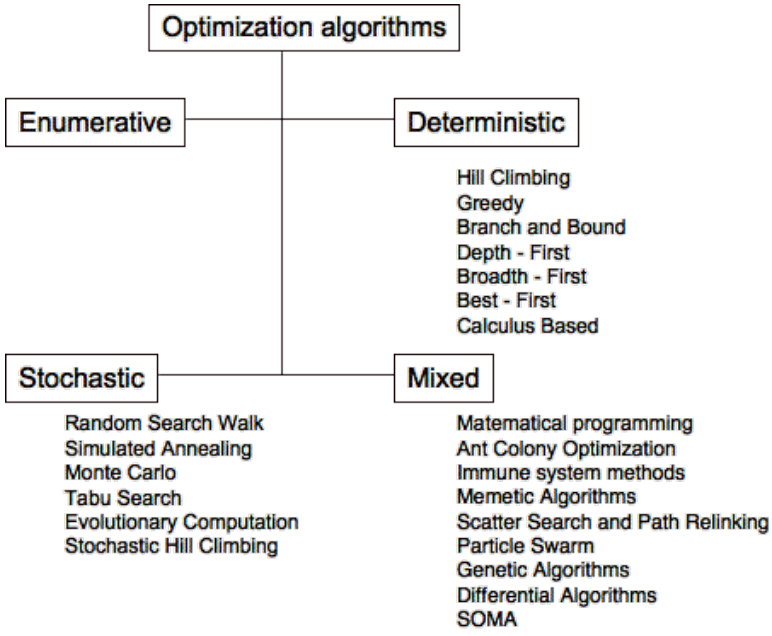


Fig. 4.1: Division of evolutionary algorithms – taken from [1]

Algorithms in stochastic and mixed group can have evolutionary features so we are talking often about evolutionary algorithms in this case. The feature are mainly in Mixed algorithms, but for e.g. Simulated Annealing can have elitism inside and then it might be called also as an evolutionary algorithm.

In the following chapters we will discuss descriptions of several evolutionary and stochastic algorithms which were used in simulations.

4.1. Deterministic algorithm – Hill Climbing

Hill Climbing (HC) algorithm belongs to one of the easiest algorithms. It searches the surface of the cost function in the direction of the biggest gradient. Therefore HC mostly ends in the nearest local optimum.

Hill Climbing starts from the randomly generated point on the surface of the cost function. Then a point from the suitable neighbourhood is chosen. Cost values of both points are compared and the point with the better value is selected as the new startpoint. The better means in the case of finding minimum – lower value, in the case of finding maximum – higher value. The first version was for finding higher value and then the point seems to climb on the hill of the cost function surface. Therefore a name Hill Climbing [9].

Other version might be that depending on the user, not only one but a certain number of points is generated in the neighbourhood. In the case that the best cost value is in the start point this one is chosen for the next loop. This subroutine is repeated several times, dependent on the user.

4.2. Stochastic algorithm - Simulated Annealing

Simulated Annealing is one of older algorithms compared to SOMA and DE. SA can be referred to as the forerunner of evolutionary algorithms [2]. It was introduced by Kirkpatrick et al. for the first time in [10]. An inspiration for developing this algorithm was annealing of metal [2], [10]. In the process metal is heated up to a temperature near melting point and then it is cooled very slowly. The purpose is to eliminate unstable particles. In other words, particles are moved towards an optimum energy state. Metal is then in a more uniform crystalline structure.

This approach was used in the case of the simulated annealing including those terms. Simulated annealing is a better variation of the Hill-Climbing algorithm [2]. Both start off from a randomly selected point. Compared to HC, simulated annealing offers a slightly different approach. It means that there is a chance to find a global optimum, not only a local one.

The principle of accepting a solution during a run of Simulated Annealing is as follows: If the new cost value is better than the old one, the new one is accepted immediately. It means that the difference between these two cost values is negative. If the difference is positive (the new cost value is worse

than the old one) a number from interval <0, 1> is generated. If it is lower than the probability according to equation (4.1), the new point is accepted, otherwise the old one continues in the process. This is called the Metropolis criterion [2], [10].

$$p(T) = e^{-\frac{\Delta E}{T}} \tag{4.1}$$

where

p(T)	- probability of transition for temperature T
ΔE	- difference between cost values of previous and current solution
T	- current temperature – control parameter for cooling schedule

The algorithm starts with high temperature T, which is decreased in steps. Equation (4.2) shows standard cooling function [2], [10].

$$T_{n+1} = \alpha\, T_n \tag{4.2}$$

where

T_{n+1}	- temperature in the next step
T_n	- temperature in the current step
α	- cooling coefficient from interval <0, 1>

Simulated Annealing offers finding a global optimum better than Hill-Climbing because probability causes that also a worse solution than the previous can be accepted, which can mean finding a global optimum in the end. Hill-Climbing goes from a start point in the direction of the biggest gradient.

4.3. Genetic Algorithms

Genetic algorithms are a group of methods which are used to solve search and optimisation problems. The basics of GA were laid down in 1975 by John H. Holland [37].

Genetic algorithms are based on natural principles of evolution, which were described by Charles Darwin. Many terms of natural genetics are also used in

genetic algorithms. Genetic algorithms work with the population of individuals. However, they work with parameters in binary code. According to the fitness function, which represents degree of quality, parents are chosen.

4.3.1. Coding and fitness function

As mentioned above, each individual has parameters called genes, which is then coded in binary form. All genes of one individual give a string called chromosome. In genetic terms, the set of parameters represented by a particular chromosome is referred to as a genotype. The genotype contains the information required to construct an organism which is referred to as the phenotype. The same terms are used in GA. Chromosome is the genotype and its cost value is phenotype [37] - [39]. The cost value says how successful the solution is. Another term connected with GA is a fitness function with the fitness value. This is used to choose parents because it says how much a particular solution is suitable. Kvasnička [2] defined the fitness as

$$F(ind) = \frac{F_{\max} - F_{\min}}{f_{\min} - f_{\max}} f(ind) + \frac{f_{\min} F_{\min} - f_{\max} F_{\max}}{f_{\min} - f_{\max}} \tag{4.3}$$

where

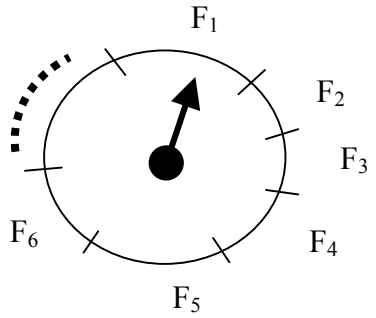
F(ind) – fitness value for an individual ind
f_{min} – minimal value of cost function
f_{max} – maximal value of cost function
F_{max} – rescaled value of cost function, value 1
F_{min} – rescaled value of cost function, value 0

Equation (4.3) assigns fitness to each individual in the population linearly in interval <0, 1> according to its cost value. The minimum value (best one) has fitness 1 and maximum value (worst one) has fitness 0. Fitness represents the interval on the unit circle or unit line [2], [37] - [39].

4.3.2. Reproduction

In the first steps, reproduction requires choosing suitable parents first. It can be done quasi-randomly by means of Roulette Wheel on the unit circle or unit line mentioned above (Fig. 4.2 and Fig. 4.3). Individuals with the lower

(higher - it depends on the programmer) cost value are more likely to be selected than individuals with the higher (lower) cost value. Besides, to change finding from minimum to maximum it is enough to multiply cost function by minus one. Reselection is allowed too. Individuals with the good cost value can be selected to become parents more often. Selected parents go to the



"mating pool". In the mating pool two parents are randomly chosen to produce two offspring.

Fig. 4.2: Scheme of RouletteWheel on unit circle

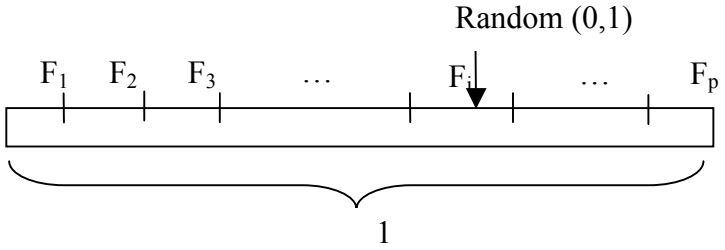


Fig. 4.3: Scheme of unit line

Their chromosomes are recombined by means of crossover and mutation [37].

Crossover means cutting of chromosomes in a randomly chosen position and changing parts between parents (see Fig. 4.4). Mostly two versions of the crossover are used; 1-point or 2-point crossover. 1-point crossover is described above; in 2-point crossover there are 2 points randomly chosen and parents change part between these two points (Fig. 4.5). Also more than 2-point crossover is possible but most often 1-point or 2-point are used because they are sufficient [37].

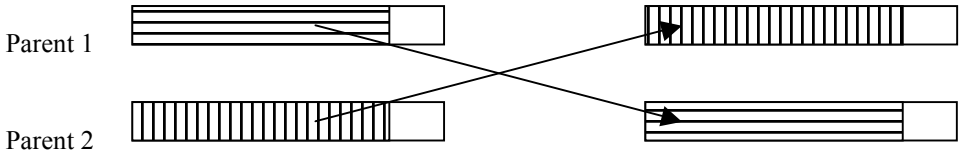


Fig. 4.4: Scheme of 1-point crossover

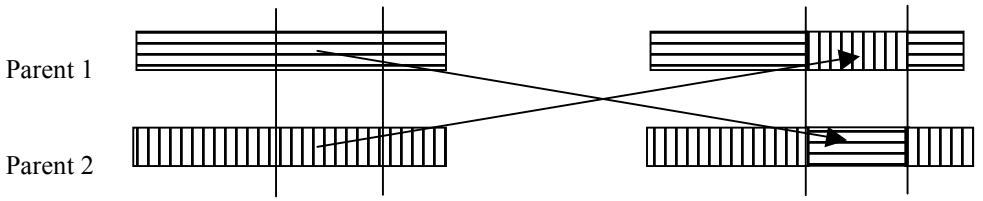


Fig. 4.5: Scheme of 2-point crossover

After crossover, each offspring goes to the process of mutation. One bit is mutated, i.e. if there is zero in the bit it is changed to one, and vice-versa. One- (Fig. 4.6) or two-bit mutation is recommended.

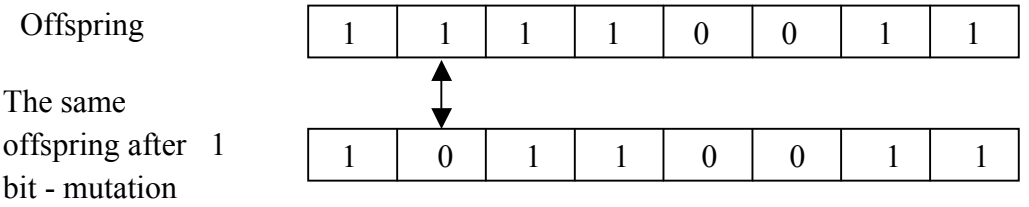


Fig. 4.6: Scheme of 1-bit mutation

4.4. Self-Organizing Migrating Algorithm (SOMA)

SOMA has been in existence since 1999 and was developed by Ivan Zelinka [5]. SOMA works with groups of individuals (population) whose

behaviour can be described as a competitive – cooperative strategy. The construction of a new population of individuals is not based on evolutionary principles (two parents produce offspring) but on the behaviour of social group, e.g. a herd of animals looking for food. This algorithm can be classified as an algorithm of social environment [45]. To the same group of algorithms particle swarm algorithm can also be put in, sometimes called swarm intelligence [6]. In the case of SOMA, no velocity vector works as in particle swarm algorithm, only the position of individuals in the search space is changed [5] during one generation, here called 'Migration loop'.

The rules are as follows: In every migration loop the best individual is chosen, i.e. individual with the minimum cost value, who is called Leader. An active individual from the population moves in the direction to Leader in the search space. At the end of the movement the position of the individual with minimum cost value is chosen. If the cost value of the new position is better than the cost value of an individual from the old population, the new one appears in new population. Otherwise the old one remains there. The movement is described by equation (4.4) and graphical explanation can be seen in Fig. 4.7.

$$x_{i,j}^{ML+1} = x_{i,j,START}^{ML} + (x_{L,j}^{ML} - x_{i,j,START}^{ML}) * t * PRTVector_j \qquad (4.4)$$

where

$x_{i,j}^{ML+1}$ - value of i–individual's j–parameter, in step t in next migration loop ML + 1

$x_{i,j,START}^{ML}$ - value of i–individual's j-parameter, Start position in actual migration loop

$x_{L,j}^{ML}$ - value of Leader's j– parameter in migration loop ML

t - step $\in$ <0, by Step to, PathLength>

PRTVector - is vector of ones and zeros depended on PRT. If random number from interval <0, 1> is less than PRT, then 1 is saved to PRTVector, otherwise it is 0.

There exists four versions of SOMA – AllToOne, AllToOneRand, AllToAll, AllToAllAdaptive. In this work we use version AllToOne despite the fact that AllToAll and AllToAllAdaptive can be much better in searching. They can search a wider area of solutions and the possibility of finding the

global optimum is then more probable. On the other hand, these two variations of SOMA need more time for its successful end of evolution. Therefore for simulations, less time-consuming computing of AllToOne was used in this work. More details can be found in [1], [5].

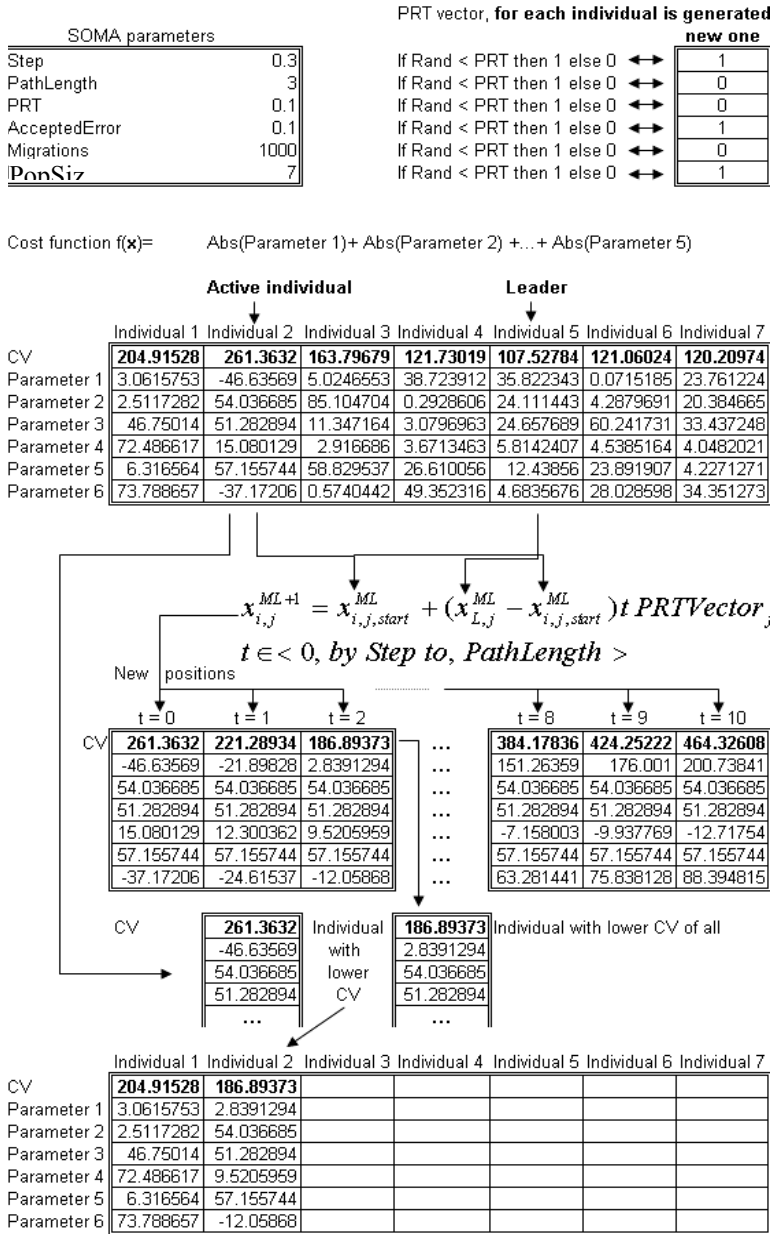


Fig. 4.7: SOMA example

4.5. Differential Evolution (DE)

Differential Evolution has been known in the scientific world since 1995. Fathers of Differential Evolution are Ken Price and Rainer Storm [44], [4].

Differential Evolution is robust, fast, and effective with global optimization ability. It does not require that the objective function is differentiable , and it works with noisy, epistatic and time-dependent objective functions.

It is a population-based optimization method that works on real-number coded individuals [44], [4]. This algorithm works also with population of individuals but there is one exception compared to other evolution algorithms. Four parents are used to produce offspring, not only two parents as is usual. For each individual $\vec{x}_{i,G}$ in the current generation *G*, DE generates a new trial individual $\vec{x}'_{i,G}$ by adding the weighted difference between two randomly selected individuals $\vec{x}_{r1,G}$ and $\vec{x}_{r2,G}$ to a third randomly selected individual $\vec{x}_{r3,G}$. The resulting individual $\vec{x}'_{i,G}$ is crossed-over with the original individual $\vec{x}_{i,G}$. The fitness of the resulting individual, referred to as perturbated vector $\vec{u}_{i,G+1}$, is then compared with the fitness of $\vec{x}_{i,G}$. If the fitness of $\vec{u}_{i,G+1}$ is greater than the fitness of $\vec{x}_{i,G}$, $\vec{x}_{i,G}$ is replaced with $\vec{u}_{i,G+1}$, otherwise $\vec{x}_{i,G}$ remains in the population as $\vec{x}_{i,G+1}$. All these actions are repeated in each generation to find the best solution. More details can be found in [44], [4].

The behaviour can be seen in detail in Fig. 4.8 [1].

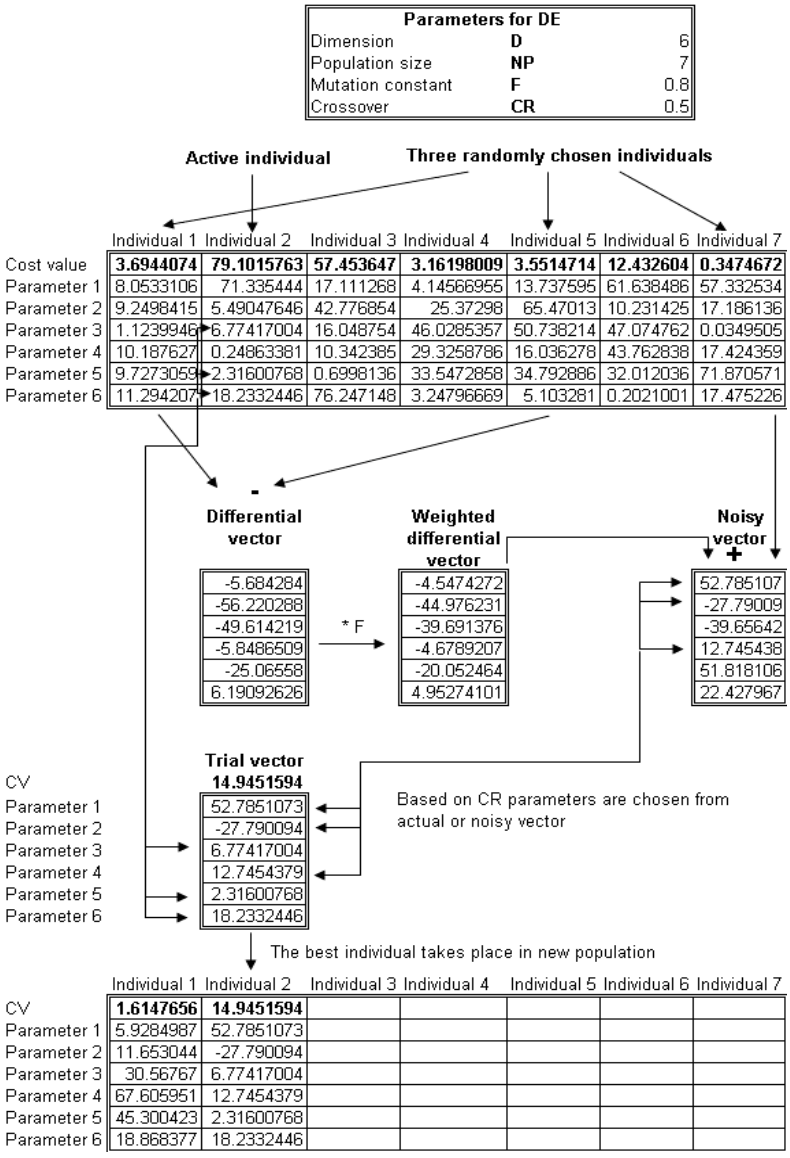


Fig. 4.8: DE example

PRACTICAL PART

5. EXPERIMENTS PERFORMED BY ANALYTIC PROGRAMMING

Several simulation experiments with Analytic Programming were done over the last four years. They were of a different nature, essentially in order to prove that AP is able to solve problems in different fields. Firstly, an approximation of data was tested, then design of electronic circuits and also settings of robot trajectory were carried out.

5.1. Data approximation

Simulations with regression were carried out on four selected problems – Quintic, Sextic, ThreeSine and FourSine problems [20]. These problems were selected from Koza's Genetic Programming [12] to compare these two methods. The aim was to find a suitable mathematical formula which fits measured data (generated from the defined functions) as well as possible.

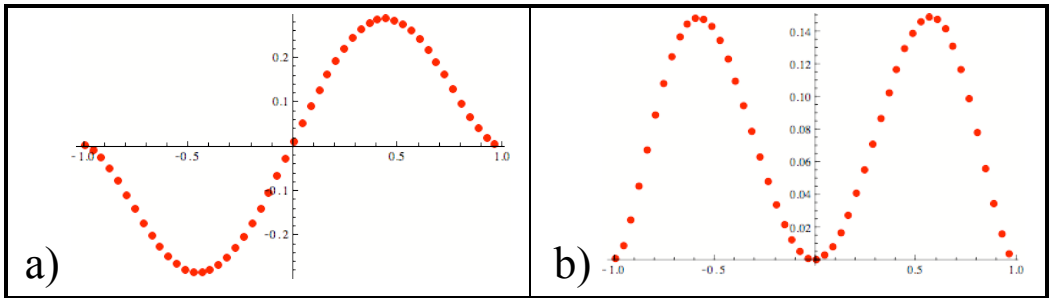
The following equations and figures show the four problems mentioned above in a practical way. Equations (5.1) - (5.4) are for Quintic, Sextic, Three Sine and Four Sine problems. The corresponding figures are given in Fig. 5.1.

$$x^5 - 2x^3 + x \tag{5.1}$$

$$x^6 - 2x^4 + x^2 \tag{5.2}$$

$$\mathrm{Sin}(x) + \mathrm{Sin}(2x) + \mathrm{Sin}(3x) \tag{5.3}$$

$$\mathrm{Sin}(x) + \mathrm{Sin}(2x) + \mathrm{Sin}(3x) + \mathrm{Sin}(4x) \tag{5.4}$$



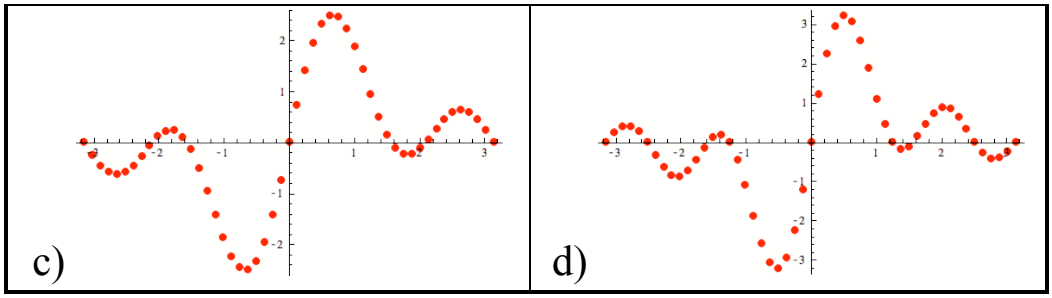


Fig. 5.1: Four problems – a) Quintic, b) Sextic polynomial in the interval [-1.0, +1.0],
c) Three Sine, d) Four Sine problem in the interval [-π, +π]

Fig. 5.2 shows examples of all 50 simulations in one picture for SOMA algorithm simulations. The nonlinearities in figures c) and d) are caused by measurement in some points not in the interval continuously.

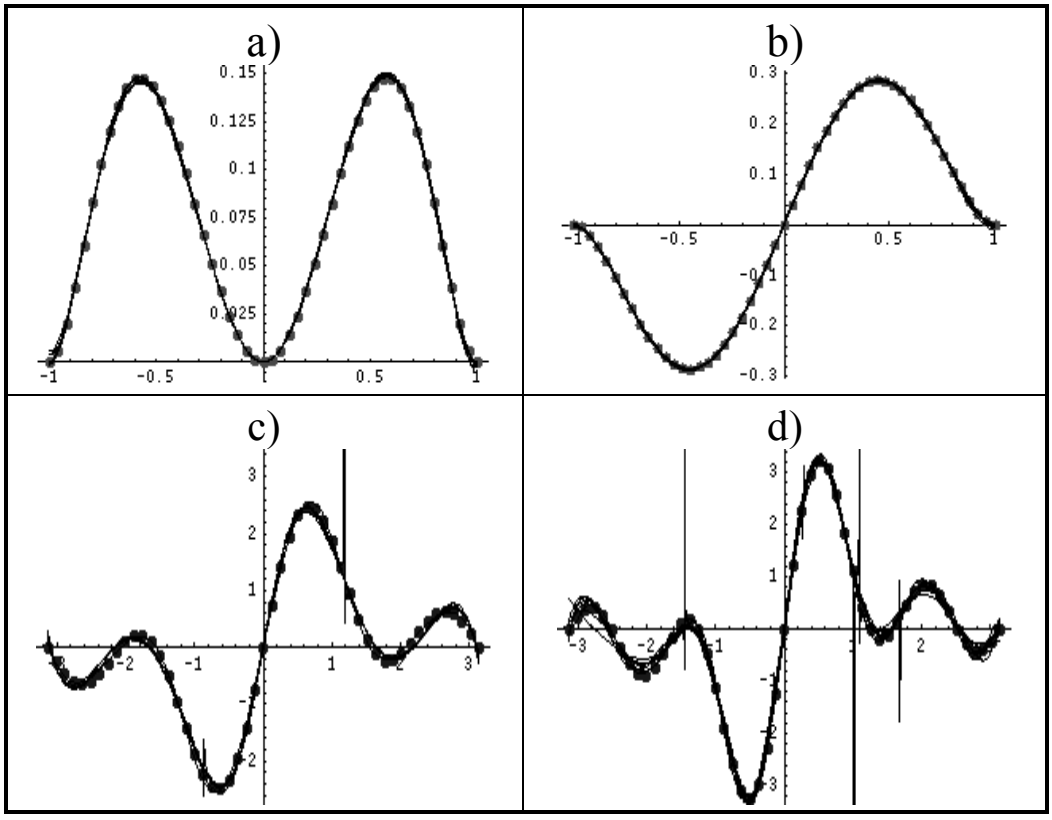


Fig. 5.2: Examples of results – a) Quintic, b) Sextic polynomial, c) Three Sine, d) Four Sine problem for SOMA algorithm

During these tests, four evolutionary algorithms were used – SOMA, GA, DE, and SA. All simulations were done 50 times. It means that 4 problems times 4 algorithms times 50 repetitions times 500 to 18 000 cost function evaluations were carried out in total.

Simulations showed that AP is faster than GP according to number of cost functions evaluations. While GP needed between 1 500 000 and 3 000 000 cost function evaluations (CFE), AP was successful with only 500 to 18 000 CFE as can be seen in Table 5.1. More details can be found in [20].

Table 5.1: Overview of results for approximation data

	GP	AP			
	GA	GA	SOMA	DE	SA
Number of individuals in population	4000	40 (40)	50 (150)	50 (150)	
Dimension of an individual		100 (120)	100 (150)	100 (150)	40 (120)
Number of evalutations of cost function	<1500 000, 3000 000>	<500, 18 000>	<500, 18 000>	<500, 18 000>	<500, 18 000>

5.2. Logical circuits design

Also in this case we chose some examples from Koza's Genetic Programming [12] – Boolean k-symmetry and even-k-parity problem. The aim was to find a suitable shape of circuits which would behave according to a given truth table. The even-k-parity problem means that the number of k

inputs with value true is even. Even-3, 4, 5, 6-parity problems were carried out. For the symmetry problems the situation is different in that the true values in inputs should be symmetric. Also for k - symmetry problems we did simulations where k was 3, 4, 5 and 6 [12, 13]. Because of the dimensions of the truth tables, there are only 3 – parity (Table 5.2) and 3 – symmetry problems (Table 5.3) for illustration.

Table 5.2: Truth table of 3- parity problem

INPUT A	INPUT B	INPUT C	OUTPUT
True	True	True	False
True	True	False	True
True	False	True	True
False	True	True	True
True	False	False	False
False	True	False	False
False	False	True	False
False	False	False	True

Table 5.3: Truth table of 3- symmetry problem

INPUT A	INPUT B	INPUT C	OUTPUT
True	True	True	True
True	True	False	False
True	False	True	True
False	True	True	False
True	False	False	False
False	True	False	True
False	False	True	False
False	False	False	True

Final output of the 3 parity problem is e.g. in (5.5) which has its earlier extended version in (5.6).

$$\neg C \wedge B \wedge A \vee \neg B \wedge C \wedge A \vee \neg A \wedge C \wedge B \vee \neg C \wedge \neg B \wedge \neg A \tag{5.5}$$

In this case also 50 simulations for each configuration of k were carried out for symmetry and parity problems. For k = 3-symmetry problem the smallest number of cost function evaluations (CFE) was 79 for DE and the highest 14 991 for SOMA algorithm in all 50 simulations for each algorithm. With increasing value of k the CFE was also increasing. For k = 6 the average value of CFE was around 200 000. The values were similar for even – k – parity problems. During simulations 49 or 50 out of 50 simulations were successful for different algorithms. More details can be found in [22].

$$
\begin{aligned}
&((C \barwedge (((A \vee C) \veebar B) \barwedge C \vee (((((B \wedge A) \barwedge (A \vee A)) \wedge (B \vee C) \veebar A) \veebar (B \veebar C)) \veebar A))) \barwedge \\
&\quad ((C \barwedge (((A \vee C) \veebar B) \barwedge C \vee (((((B \wedge A) \barwedge (A \vee A)) \wedge (B \vee C) \veebar A) \veebar (B \veebar C)) \veebar A))) \barwedge ((A \vee B) \barwedge B) \veebar \\
&\quad\quad (A \vee (C \barwedge C \veebar B \wedge (C \barwedge (((A \vee C) \veebar B) \barwedge C \vee (((((B \wedge A) \barwedge (A \vee A)) \wedge (B \vee C) \veebar A) \veebar (B \veebar C)) \veebar A)))) \wedge \\
&\quad\quad\quad (C \barwedge (((A \vee C) \veebar B) \barwedge C \vee (((((B \wedge A) \barwedge (A \vee A)) \wedge (B \vee C) \veebar A) \veebar (B \veebar C)) \veebar A)) \veebar B) \wedge \\
&\quad\quad\quad (C \barwedge (((A \vee C) \veebar B) \barwedge C \vee (((((B \wedge A) \barwedge (A \vee A)) \wedge (B \vee C) \veebar A) \veebar (B \veebar C)) \veebar A))) \wedge (B \vee B) \vee C)) \vee \\
&\quad (C \barwedge (((A \vee C) \veebar B) \barwedge C \vee (((((B \wedge A) \barwedge (A \vee A)) \wedge (B \vee C) \veebar A) \veebar (B \veebar C)) \veebar A)) \veebar \\
&\quad\quad ((C \barwedge (((A \vee C) \veebar B) \barwedge C \vee (((((B \wedge A) \barwedge (A \vee A)) \wedge (B \vee C) \veebar A) \veebar (B \veebar C)) \veebar A))) \barwedge ((A \vee B) \barwedge B) \veebar \\
&\quad\quad\quad (A \vee (C \barwedge C \veebar B \wedge (C \barwedge (((A \vee C) \veebar B) \barwedge C \vee (((((B \wedge A) \barwedge (A \vee A)) \wedge (B \vee C) \veebar A) \veebar (B \veebar C)) \veebar A)))) \wedge \\
&\quad\quad\quad\quad (C \barwedge (((A \vee C) \veebar B) \barwedge C \vee (((((B \wedge A) \barwedge (A \vee A)) \wedge (B \vee C) \veebar A) \veebar (B \veebar C)) \veebar A)) \veebar B) \wedge \\
&\quad\quad\quad\quad (C \barwedge (((A \vee C) \veebar B) \barwedge C \vee (((((B \wedge A) \barwedge (A \vee A)) \wedge (B \vee C) \veebar A) \veebar (B \veebar C)) \veebar A))) \wedge (B \vee B) \vee C)))) \wedge \\
&(C \barwedge \\
&\quad ((B \vee \\
&\quad\quad (((C \barwedge (((A \vee C) \veebar B) \barwedge C \vee (((((B \wedge A) \barwedge (A \vee A)) \wedge (B \vee C) \veebar A) \veebar (B \veebar C)) \veebar A))) \barwedge ((A \vee B) \barwedge B) \veebar \\
&\quad\quad\quad (A \vee (C \barwedge C \veebar B \wedge (C \barwedge (((A \vee C) \veebar B) \barwedge C \vee (((((B \wedge A) \barwedge (A \vee A)) \wedge (B \vee C) \veebar A) \veebar (B \veebar C)) \veebar A)))) \wedge \\
&\quad\quad\quad\quad (C \barwedge (((A \vee C) \veebar B) \barwedge C \vee (((((B \wedge A) \barwedge (A \vee A)) \wedge (B \vee C) \veebar A) \veebar (B \veebar C)) \veebar A)) \veebar B) \wedge \\
&\quad\quad\quad\quad (C \barwedge (((A \vee C) \veebar B) \barwedge C \vee (((((B \wedge A) \barwedge (A \vee A)) \wedge (B \vee C) \veebar A) \veebar (B \veebar C)) \veebar A))) \wedge (B \vee B) \vee C)) \vee \\
&\quad\quad C \vee (A \veebar A)) \wedge (A \barwedge C \vee C \vee B)) \wedge \\
&\quad\quad (((C \barwedge (((A \vee C) \veebar B) \barwedge C \vee (((((B \wedge A) \barwedge (A \vee A)) \wedge (B \vee C) \veebar A) \veebar (B \veebar C)) \veebar A))) \barwedge A \vee C) \barwedge \\
&\quad\quad (C \wedge \\
&\quad\quad\quad (((A \veebar (B \vee B \wedge ((C \barwedge (((A \vee C) \veebar B) \barwedge C \vee (((((B \wedge A) \barwedge (A \vee A)) \wedge (B \vee C) \veebar A) \veebar (B \veebar C)) \veebar A))) \barwedge ((A \vee B) \barwedge B) \veebar \\
&\quad\quad\quad\quad\quad (A \vee (C \barwedge C \veebar B \wedge (C \barwedge (((A \vee C) \veebar B) \barwedge C \vee (((((B \wedge A) \barwedge (A \vee A)) \wedge (B \vee C) \veebar A) \veebar (B \veebar C)) \veebar \\
&\quad\quad\quad\quad\quad\quad A)))) \wedge (C \barwedge (((A \vee C) \veebar B) \barwedge C \vee \\
&\quad\quad\quad\quad\quad\quad\quad (((((B \wedge A) \barwedge (A \vee A)) \wedge (B \vee C) \veebar A) \veebar (B \veebar C)) \veebar A)) \veebar B) \wedge \\
&\quad\quad\quad\quad\quad (C \barwedge (((A \vee C) \veebar B) \barwedge C \vee (((((B \wedge A) \barwedge (A \vee A)) \wedge (B \vee C) \veebar A) \veebar (B \veebar C)) \veebar A))) \wedge \\
&\quad\quad\quad\quad\quad (B \vee B) \vee C)))) \wedge C \wedge (A \veebar B) \veebar \\
&\quad\quad\quad (((C \barwedge (((A \vee C) \veebar B) \barwedge C \vee (((((B \wedge A) \barwedge (A \vee A)) \wedge (B \vee C) \veebar A) \veebar (B \veebar C)) \veebar A))) \barwedge ((A \vee B) \barwedge B) \veebar \\
&\quad\quad\quad\quad (A \vee (C \barwedge C \veebar B \wedge (C \barwedge (((A \vee C) \veebar B) \barwedge C \vee (((((B \wedge A) \barwedge (A \vee A)) \wedge (B \vee C) \veebar A) \veebar (B \veebar C)) \veebar A)))) \wedge \\
&\quad\quad\quad\quad\quad (C \barwedge (((A \vee C) \veebar B) \barwedge C \vee (((((B \wedge A) \barwedge (A \vee A)) \wedge (B \vee C) \veebar A) \veebar (B \veebar C)) \veebar A)) \veebar B) \wedge \\
&\quad\quad\quad\quad\quad (C \barwedge (((A \vee C) \veebar B) \barwedge C \vee (((((B \wedge A) \barwedge (A \vee A)) \wedge (B \vee C) \veebar A) \veebar (B \veebar C)) \veebar A))) \wedge (B \vee B) \vee \\
&\quad\quad\quad\quad\quad C)) \vee A \vee A \vee A) \wedge ((B \veebar A) \veebar C \barwedge B)) \barwedge C))))) \wedge \\
&\quad (A \barwedge \\
&\quad\quad ((B \vee \\
&\quad\quad\quad (((C \barwedge (((A \vee C) \veebar B) \barwedge C \vee (((((B \wedge A) \barwedge (A \vee A)) \wedge (B \vee C) \veebar A) \veebar (B \veebar C)) \veebar A))) \barwedge ((A \vee B) \barwedge B) \veebar \\
&\quad\quad\quad\quad (A \vee (C \barwedge C \veebar B \wedge (C \barwedge (((A \vee C) \veebar B) \barwedge C \vee (((((B \wedge A) \barwedge (A \vee A)) \wedge (B \vee C) \veebar A) \veebar (B \veebar C)) \veebar A)))) \wedge \\
&\quad\quad\quad\quad\quad (C \barwedge (((A \vee C) \veebar B) \barwedge C \vee (((((B \wedge A) \barwedge (A \vee A)) \wedge (B \vee C) \veebar A) \veebar (B \veebar C)) \veebar A)) \veebar B) \wedge \\
&\quad\quad\quad\quad\quad (C \barwedge (((A \vee C) \veebar B) \barwedge C \vee (((((B \wedge A) \barwedge (A \vee A)) \wedge (B \vee C) \veebar A) \veebar (B \veebar C)) \veebar A))) \wedge (B \vee B) \vee C)) \vee \\
&\quad\quad\quad C \vee (A \veebar A)) \wedge (A \barwedge C \vee C \vee B)) \wedge \\
&\quad\quad\quad (((C \barwedge (((A \vee C) \veebar B) \barwedge C \vee (((((B \wedge A) \barwedge (A \vee A)) \wedge (B \vee C) \veebar A) \veebar (B \veebar C)) \veebar A))) \barwedge A \vee C) \barwedge \\
&\quad\quad\quad\quad (C \wedge \\
&\quad\quad\quad\quad\quad (((A \veebar (B \vee B \wedge ((C \barwedge (((A \vee C) \veebar B) \barwedge C \vee (((((B \wedge A) \barwedge (A \vee A)) \wedge (B \vee C) \veebar A) \veebar (B \veebar C)) \veebar A))) \barwedge ((A \vee B) \barwedge B) \veebar \\
&\quad\quad\quad\quad\quad\quad\quad (A \vee (C \barwedge C \veebar B \wedge (C \barwedge (((A \vee C) \veebar B) \barwedge C \vee (((((B \wedge A) \barwedge (A \vee A)) \wedge (B \vee C) \veebar A) \veebar (B \veebar C)) \veebar \\
&\quad\quad\quad\quad\quad\quad\quad\quad\quad A)))) \wedge (C \barwedge (((A \vee C) \veebar B) \barwedge C \vee \\
&\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad (((((B \wedge A) \barwedge (A \vee A)) \wedge (B \vee C) \veebar A) \veebar (B \veebar C)) \veebar A)) \veebar B) \wedge \\
&\quad\quad\quad\quad\quad\quad\quad\quad\quad (C \barwedge (((A \vee C) \veebar B) \barwedge C \vee (((((B \wedge A) \barwedge (A \vee A)) \wedge (B \vee C) \veebar A) \veebar (B \veebar C)) \veebar A))) \wedge \\
&\quad\quad\quad\quad\quad\quad\quad\quad\quad (B \vee B) \vee C)))) \wedge C \wedge (A \veebar B) \veebar \\
&\quad\quad\quad\quad\quad\quad\quad (((C \barwedge (((A \vee C) \veebar B) \barwedge C \vee (((((B \wedge A) \barwedge (A \vee A)) \wedge (B \vee C) \veebar A) \veebar (B \veebar C)) \veebar A))) \barwedge ((A \vee B) \barwedge B) \veebar \\
&\quad\quad\quad\quad\quad\quad\quad\quad (A \vee (C \barwedge C \veebar B \wedge (C \barwedge (((A \vee C) \veebar B) \barwedge C \vee (((((B \wedge A) \barwedge (A \vee A)) \wedge (B \vee C) \veebar A) \veebar (B \veebar C)) \veebar A)))) \wedge \\
&\quad\quad\quad\quad\quad\quad\quad\quad\quad (C \barwedge (((A \vee C) \veebar B) \barwedge C \vee (((((B \wedge A) \barwedge (A \vee A)) \wedge (B \vee C) \veebar A) \veebar (B \veebar C)) \veebar A)) \veebar B) \wedge \\
&\quad\quad\quad\quad\quad\quad\quad\quad\quad (C \barwedge (((A \vee C) \veebar B) \barwedge C \vee (((((B \wedge A) \barwedge (A \vee A)) \wedge (B \vee C) \veebar A) \veebar (B \veebar C)) \veebar A))) \wedge (B \vee B) \vee \\
&\quad\quad\quad\quad\quad\quad\quad\quad\quad C)) \vee A \vee A \vee A) \wedge ((B \veebar A) \veebar C \barwedge B)) \barwedge C))))) \veebar \\
&((B \barwedge ((C \barwedge (((A \vee C) \veebar B) \barwedge C \vee (((((B \wedge A) \barwedge (A \vee A)) \wedge (B \vee C) \veebar A) \veebar (B \veebar C)) \veebar A))) \barwedge ((A \vee B) \barwedge B) \veebar \\
&\quad\quad\quad (A \vee (C \barwedge C \veebar B \wedge (C \barwedge (((A \vee C) \veebar B) \barwedge C \vee (((((B \wedge A) \barwedge (A \vee A)) \wedge (B \vee C) \veebar A) \veebar (B \veebar C)) \veebar A)))) \wedge \\
&\quad\quad\quad\quad (C \barwedge (((A \vee C) \veebar B) \barwedge C \vee (((((B \wedge A) \barwedge (A \vee A)) \wedge (B \vee C) \veebar A) \veebar (B \veebar C)) \veebar A)) \veebar B) \wedge \\
&\quad\quad\quad\quad (C \barwedge (((A \vee C) \veebar B) \barwedge C \vee (((((B \wedge A) \barwedge (A \vee A)) \wedge (B \vee C) \veebar A) \veebar (B \veebar C)) \veebar A))) \wedge (B \vee B) \vee C)) \veebar \\
&\quad\quad\quad (C \barwedge (((A \vee C) \veebar B) \barwedge C \vee (((((B \wedge A) \barwedge (A \vee A)) \wedge (B \vee C) \veebar A) \veebar (B \veebar C)) \veebar A)) \veebar \\
&\quad\quad\quad\quad C \barwedge (((A \vee C) \veebar B) \barwedge C \vee (((((B \wedge A) \barwedge (A \vee A)) \wedge (B \vee C) \veebar A) \veebar (B \veebar C)) \veebar A)))) \vee (B \veebar A) \wedge C)
\end{aligned}
\tag{5.6}
$$

5.3. Optimal setting of robot trajectory

This task was used to prove that AP is able to work also with linguistic terms which in real words means some commands for robot like move straight-forwardly, turn left, turn right, look before and find what is there, etc. [23], [24]. To try to see, if it works, we chose a task Santa Fe Trail for artificial ant from Koza's Genetic Programming [12].

The problem was designed so that an artificial ant should go through a defined trail (Fig. 5.3) and eat all the food that was there. The trail was set up as 31 x 32 fields where black field means food, grey and white is basically the same, i.e. there is nothing. But the grey colour was used to highlight the problems on the way for the ant which are e.g.:

- one simple hole (position [8,27] in Fig. 5.3)
- two holes in the line (positions [13,16] and [13,17])
- three holes ([17,15], [17,16], [17,17])
- holes in the corners
    - one hole (position [13,8]]
    - two holes ([1,8],[2,8])
    - three holes ([17,15], [17,16], [17,17])

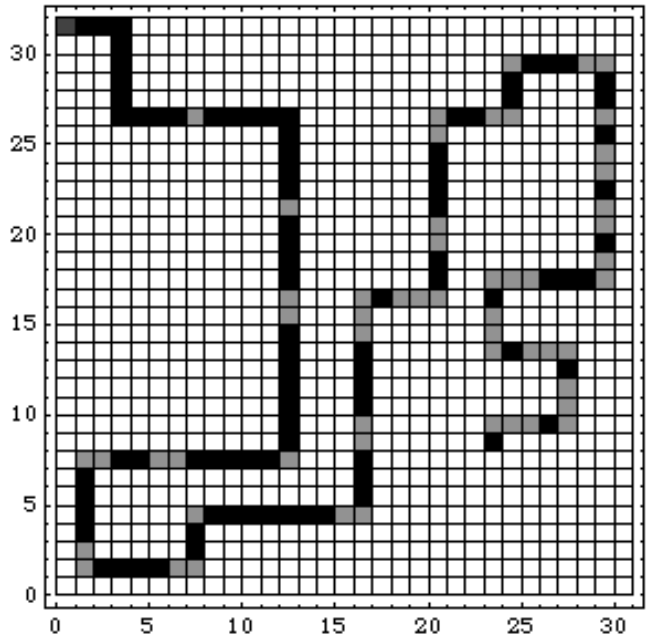


Fig. 5.3: Santa Fe Trail for artificial ant

The cost function was set up as in (5.7)

$$CV = 89 - NumberFood \tag{5.7}$$

5.3.1. Set of functions

The set of functions used for movements of the ant is following. As a set of variables GFS_{0arg} [22], i. e. functions, which provide moving of an ant, without any argument which could be add during the process of evolution.

The set consist of

- GFS0 = {Left, Right, Move},

where

GFS0 – a set of variables and terminals, zero-arguments functions GFS_{0arg} [22],

Left – function for turning around in the anticlockwise direction

Right – function for turning around in the clockwise direction

Move – function for moving straight and if a bait is in the field where the ant is moved, it is eaten.

This set of functions is not enough to make successfully a desired task. More functions are necessary. Then a GFS2 and GFS3 were set up.

- GFS2 = {IfFoodAhead, Prog2}
- GFS3 = {Prog3}

Where the number in GFS means the arity of the functions inside, i.e. number of arguments which are needed to be evaluated correctly. Arguments are added to those functions during evolution process as described in chapter about AP.

IfFoodAhead is a decision function – the ant controls the field in front of it and if there is food, the function in the field for truth argument is executed; otherwise function in false position is performed.

Prog2 and **Prog3** are the same function in the principle. They do 2 or 3 functions in the same time. These two functions were originally defined also in

Koza's approach but in AP it is necessary because of the structure of generating the program.

5.3.2. Results

50 simulations were performed for this task with three used algorithms. SOMA and DE were more successful with the same settings of cost function evaluations. SA was successful only in a third of the cases.

The solutions obtained were also under 400 steps which was the request to eat all the food as can be seen in Table 5.4. The best was 367-step-solution found by DE. The limit of 400 steps was one of conditions for cost function defined by Koza [12]. But his solution showed that it did not fit this condition at all. It needed 545 steps [46] as also our simple simulation proved.

Table 5.4: Number of steps for artificial ant

	Number of steps		
	SOMA	DE	SA
Minimum	396	367	406
Maximum	606	604	605
Average	547	540	535

This task was time consuming which means that one simulation can hypothetically take 1 – 3 days on the computer with Athlon XP 1800+ processor, 256 MB RAM memory, Windows XP and Mathematica 5.2. It depends on a number of cost function evaluations. One cost function evaluation took 1 to 6 seconds. The time could be decreased by parallelization of the process, which is one of the further plans. Koza did parallelization in

GP as well. He uses GP activity computer-cluster consisting of hundreds of PCs [47]. But in our case we used only **1** computer for all simulations.

5.3.3. Output from the simulations

The output from the simulations was the rules of how to move on the grid with the food. The winner was DE with 367 steps to reach the final field as shown in (5.8)

IfFoodAhead[Move, IfFoodAhead[Move, Prog2[Prog2[Right, IfFoodAhead[Prog2[IfFoodAhead[IfFoodAhead[Move, Move], Move], Move], Prog3[IfFoodAhead[Move, IfFoodAhead[Prog3[Right, Right, Prog2[Left, Prog2[IfFoodAhead[Prog2[Prog2[Left, Move], Right], IfFoodAhead[Move, Left]], Prog2[IfFoodAhead[Move, Move], Prog2[IfFoodAhead[Move, Right], Right]]]]], Left]], Left, IfFoodAhead[Move, Right]]]], Move]]] (5.8)

This system of rules shows how the ant should behave on its way. This does not solve the concrete trajectory step by step. These rules should work also on other similar types of grids where the condition concerned to food is applied, i.e. food must be in the neighbourhood lines. There must not be a free line between them. The previously described problems might appear even in bigger number of holes in the same line. If we change the grid and placement of the food the number of steps will obviously change. Details can be seen in [23], [24].

5.4. Local conclusion and discussion

The previous three case studies showed that AP is able to work with numerical values, approximation data as well as with linguistic terms which were either operators for design of electronic circuits and setting of optimal trajectory for a robot. This presumptions leads to the conclusion that AP

should be able to work also with simple operators of evolutionary algorithms and create a new one.

The proof was also given in synthesis of chaos [26] and neural networks [25].

6. CREATION OF EVOLUTIONARY ALGORITHMS - PROGRESS

6.1. First experiments

The objective was to try to create a new optimization algorithm, probably of evolutionary character, which could be robust and effective to optimize difficult problems in the world. This is a metaevolution in the context we use it. According to previous approaches, metaevolution is determining the optimal evolutionary algorithm, best types of evolutionary operator and their parameter setting for a given problem. It means basically, that one evolutionary algorithm tunes another one [9]. But our approach is different. We use metaevolution on higher level for creating a new algorithm completely, not only for setting of its parameters [11].

The research started with collecting of operators of known evolutionary algorithms like mutation or crossover. These operators are used as basic simple functions for Analytic Programming.

The first step was to try to create an algorithm which is known from its basic operators. Differential Evolution was used because of its simple structure and its easy implementation.

The original algorithm of DERand1Bin version of Differential Evolution [44], [4] is described in the section about evolutionary algorithms.

For the purposes of creating DE back we extracted the following operators.

SelectionDE – this is operator which chooses individuals from population for other instructions. In this case the output will be four individuals – one active individual and 3 randomly chosen ones.

MutationDE - into this operator four individuals are coming from SelectionDE. Here mutation is produced as follows: one of the randomly chosen parents is subtracted from the second parent and so called differential vector is produced. This one is multiplied by a mutation constant and the result of this operation is a weighted differential vector. The third parent plus the weighted differential vector give a noise vector. The noise vector is the output of the MutationDE.

CrossoverDE – the active individual provides some arguments and the input individual to CrossoverDE gives some other arguments and the trial vector is also given.

All operators are then applied on each individual in a sequence. The last operator which produces the complete algorithm is called FinalDE. Inside of this the output individual is compared with the active one and the one with better or the same value of cost function appears in the new population.

The original Differential Evolution then appears like in the equation (6.1) consisting from the above described operators:

$$\text{CrossoverDE}(\text{MutationDE}(\text{SelectionDE}))) \tag{6.1}$$

This means in words, that firstly operator SelectionDE will be used. Its result will be put inside the operator MutationDE and at the end CrossoverDE will be applied. The algorithm DE is described also in section about evolutionary algorithms in detail. The inscription is also able to write as (6.2). But for further simulations we used type (6.1).

$$\text{CrossoverDE} \circ \text{MutationDE} \circ \text{SelectionDE} \tag{6.2}$$

Analytic Programming was then applied on DE operators. Parameter setting for AP is described in the next section.

6.1.1. General Function Set

Analytic Programming produces a final formula from elementary functions. In this case elementary functions are DE operators described above. In AP, subsets of simple functions according to the number of arguments are called GFS_{0arg}, $GFS_{1arg,}GFS_{2arg}$ etc. GFS stands for General Function Set which was described in [18] - [24].

The operators of DE are put into these subsets as follows.

GFS_{0arg} - Set of 0-arguments functions, so called terminals, contains SelectionDE which produces 4 individuals.

GFS_{1arg} - Set of 1-arguments functions contains MutationDE and CrossoverDE. Both need to have some individuals produced e.g. from SelectDE as input.

6.1.2. Cost Function

When Analytic Programming creates a complex formula, it is necessary to assign some value that represents the suitability of the individual and its quality. In the case of creating a new evolutionary algorithm, benchmarking on some test functions is necessary. In the first preliminary study we concentrated to try the just generated algorithm on two test functions - whether it achieves the minimum in both test functions or not. Two cost functions were Sphere model, 1st De Jong as example of unimodal function and Schwefel as example of multimodal function [1] – Fig. 6.1 and Fig. 6.2.

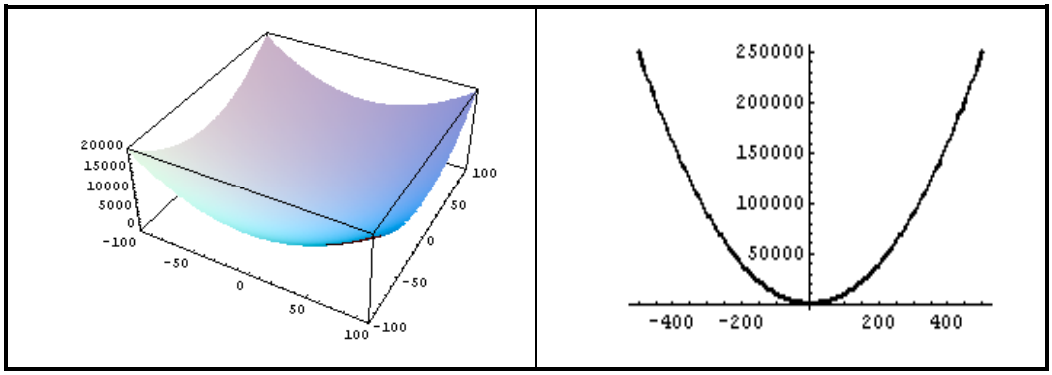


Fig. 6.1: DeJong function – unimodal (left – 2 arguments and right – 1 argument used)

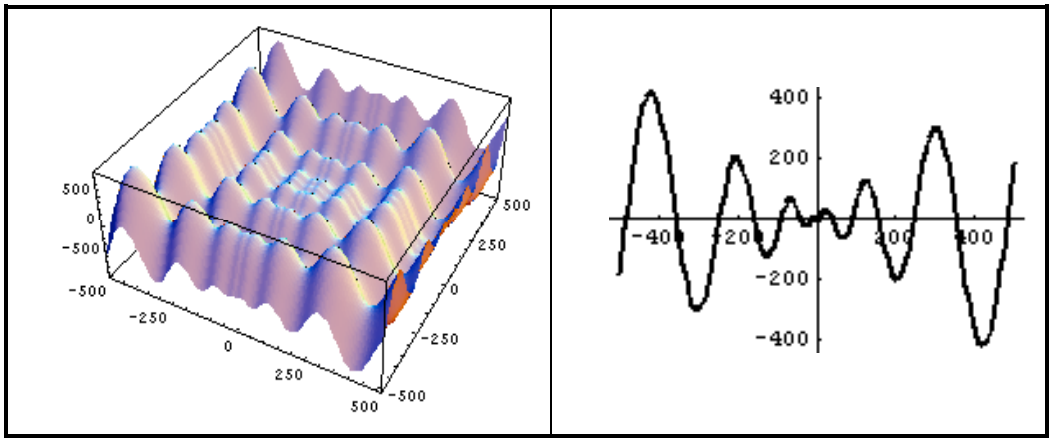


Fig. 6.2: Schwefel function – multimodal (left – 2 arguments and right – 1 argument used)

1st De Jong and Schwefel functions are in analytical way as seen in equations (6.3) and (6.4). No other condition was applied.

$$f(x) = \sum_{i=1}^{Dim} x_i^2 \tag{6.3}$$

$$f(x) = \sum_{i=1}^{Dim} -x_i \cdot \sin\left(\sqrt{|x_i|}\right) \tag{6.4}$$

The value of the cost function was designed so that firstly the generated algorithm is verified as to the ability to find minimum on the easy unimodal function 1st De Jong. If the minimum is reached the Schwefel function is tested. Then the cost value is the output from Schwefel. If there is no successful result from 1st De Jong, the output value is the absolute value of 1st De Jong. The values were known because we used test functions with 2 arguments.

6.1.3. Results of the preliminary study

The aim of the preliminary study and the aim of these simulations was to find the original Differential Evolution. The settings for parameters were done according to heuristic analysis during the use of it.

The length of the individual in Analytic Programming was set up to 15 and the number of simple functions to 3. It means that there exists 32767 possibilities of generated algorithms according to variations with repetition.

And it is quite natural that also in the first population, randomly generated without any evolution, can be found algorithms which can fit the minimum. Following examples show successful and also unsuccessful individuals, i.e. generated algorithms.

Examples of generated algorithms which were not successful (6.5) - (6.7):

CrossoverDE(CrossoverDE(MutationDE(CrossoverDE(CrossoverDE(SelectionDE))))) (6.5)

SelectionDE (6.6)

CrossoverDE(SelectionDE) (6.7)

and also some successful examples (6.8) and (6.9):

CrossoverDE(MutationDE(SelectionDE)) the original Differential Evolution (6.8)

MutationDE(MutationDE(MutationDE(MutationDE(SelectionDE))) (6.9)

The final (6.9) was tried in simulations of 100 times as the original DE with the same settings of parameters as the original DE.

The following pictures Fig. 6.3 displays behaviour of algorithms during 100 repeated simulations – a) original algorithm DE and b) a newly created one.

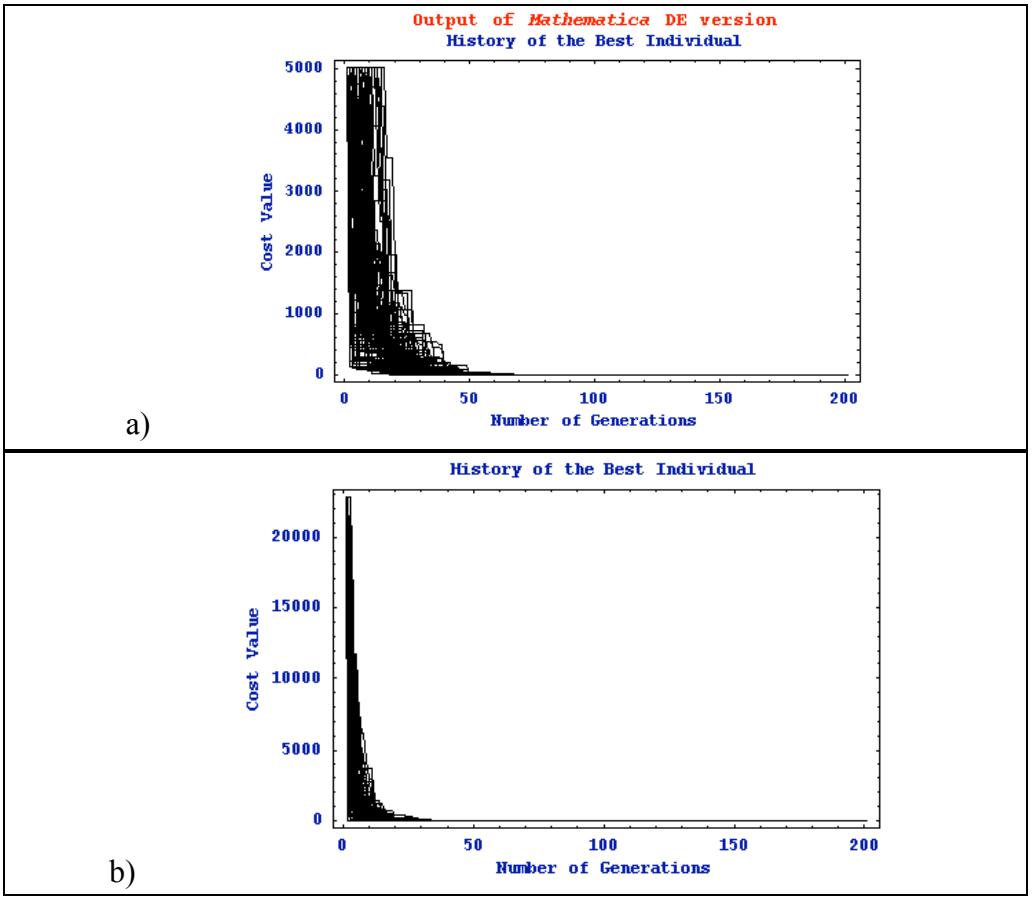


Fig. 6.3: 100 simulations for 1st De Jong a) original DE, b) new algorithm

The results are shown also for Schwefel function in Fig. 6.4. Comparison of minimal and maximal values during all 100 simulations is also in Table 6.1.

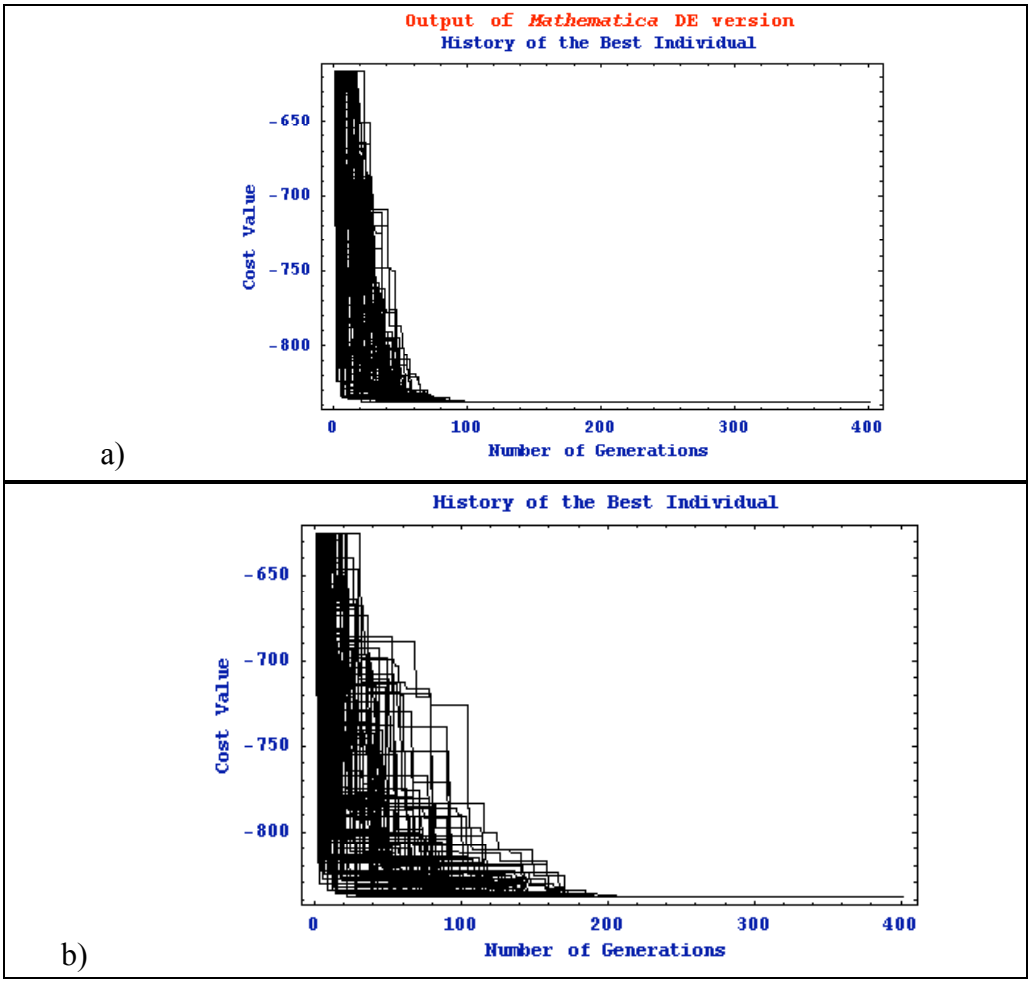


Fig. 6.4: History of best individual for Schwefel a) original DE and b) new algorithm

As can be seen, the generated program was able to find minimum as well as the original DE. It is basically the same, only there are more mutations. In the unimodal function the convergence is faster than in original DE but in the case of Schwefel function it took more time to achieve the minimum on average. The more important is faster convergence in the multimodal functions because such problems are found in every day optimization. And the requirement is to

find the optimum as fast as possible, and of course, not to finish in the local optimum.

Table 6.1: Comparison of results of original DE and generated algorithm

	ORIGINAL DE		**GENERATED ALGORITHM**	
	1st De Jong	Schwefel	1st De Jong	Schwefel
Minimum	2.04492 x 10-8	-837.966	2.39949 x 10-16	-837.966
Maximum	6.61369 x 10-6	-837.966	1.45227 x 10-14	-837.966
Average	9.29224 x 10-7	-837.966	3.6897 x 10-15	-837.966

In later analysis of the solution it was found that it is another version of the DE. There are only more individuals which are used for mutations to create a new individual for a new population.

6.2. Design of new cost function

6.2.1. New operators added and renamed

For the purpose to create evolutionary algorithms by means of Analytic Programming, we extended algorithms from Differential Evolution from the previous section and [48] also to Self-Organizing Migrating Algorithm, Hill Climbing algorithm and Simulated Annealing. Details of these algorithms can be found in [4], [5], [9] and [10]. It was necessary to separate its operators like mutation, crossover and selection of parents. The following operators were put inside GFS sets according to the number of arguments.

We had to also rename the operators since we used more versions of some algorithms. Therefore we need to distinguish between them. The name contains also connections to the appropriate versions [50].

GFS_{0arg}= {SelectDE, SelectLeaderSOMA, SelectSOMARandLeader, SelectHC}

GFS_{1arg}= { MutateDERand1, CrossDEExp, CrossDEBin, MutateDEBest2, MutateDERand2, MutateDECurrentToBest, MutateDEBest1, SOMAATOWithPRT, SOMAATOWithoutPRT, SOMAATORandWithPRT, SOMAATORandWithoutPRT, CompleteHC, CompleteSA }

SelectDE – this is the operator which selects individuals from population for other instructions. In this case, the output will be 4 individuals – one active individual and 3 randomly chosen.

MutateDERand1- here mutation is produced as follows: one of the randomly chosen parents is subtracted from the second parent and a so called differential vector is produced. This vector is multiplied by a mutable constant and the result of this operation is a weighted differential vector. The third parent plus the weighted differential vector produces a noisy vector. This noisy vector is the output of the MutateDERand1.

MutateDEBest2, MutateDEBest1, MutateDERand2, MutateDECurrentToBest are mutation functions of other version of Differential Evolution.

CrossDEBin – the active individual gives some arguments and the input individual to CrossDEBin gives some other arguments and the trial vector is created. This is given by crossover constant Cr. If random number from interval <0,1> is less than Cr the arguments from active individual is taken, otherwise it is from the individual which is input of CrossDEBin.

CrossDEExp is similar crossover to *CrossDEBin*. The difference is in the choice of arguments into the trial vector. Until first case of random number from interval <0,1> is less than Cr, arguments from active individual are taken, then the rest from the input individual of CrossDEExp [49].

SelectLeaderSOMA – chooses the best individual in the population (with the minimal value of cost function).

SelectSOMARandLeader – choses the random individual from population.

SOMAATOWithPRT – is the operator which create a table of new individuals which are in the direction from active individual to Leader in Steps and the best individual is selected as an output individual.

SOMAATOWithoutPRT, SOMAATORandWithPRT, SOMAATORandWithoutPRT – are similar as the previous one, the only difference is in the use of PRTVector and best individual as Leader or random individual as Leader.

SelectHC – chooses random point in the Cost Function.

CompleteHC – is a process of Hill Climbing algorithm. If the randomly chosen point from the neighbourhood has less cost value, it is chosen as a new startpoint, otherwise the current start point is used again.

CompleteSA – is a process of Simulated Annealing algorithm. If the randomly chosen point from the neighbourhood has less cost value, it is chosen as a new start point, otherwise the condition of probability of acceptance a worse solution is applied or the current start point is used again.

All above described operators work as modules with some input and some output. The functionality is related to one active individual. Therefore for application for all individuals in the population *FinalAlgorithm* is set up as well.

Original Differential Evolution of DERand1Bin version can be written as the equation (6.10).

CrossDEBin(MutateRand1(SelectDE)) (6.10)

Original SOMA in version All To One is then used as equation (6.11).

SOMAATOWithPRT (SelectLeaderSOMA) (6.11)

Hill Climbing has similar notation (equation(6.12))

CompleteHC (SelectHC) (6.12)

Hence, Simulated Annealing has the same method of selecting individuals at the beginning as the HC we used for the notation (6.13) same operators.

CompleteSA (SelectHC) (6.13)

6.2.2. Design of cost function

The testing functions were the same as in the previous section – 1st De Jong and Schwefel. The value of Cost Function was designed so that initially the generated algorithm is observed if it is able to find the minimum value on the easy unimodal function 1st De Jong. Better said, it is testing the difference between global extreme and the extreme approached by a new generated algorithm. If the difference under 10^{-7} is reached, then the Schwefel function is tested similarly. We change the approach to the value in order to measure the difference as the optimization might be easier in that the order of the cost value will be the same for both functions and we can easily work with the penalization without any fear whether the values of some functions are suitable or not.

If the algorithm is successful on both functions, the value is set as seen equation (6.14) in the case that number of cost function evaluations were less than the average.

$$|\text{CFESchwefel} - \text{avgCFESchwefel}| / \text{SchwefelValue} \quad (6.14)$$

where

CFESchwefel is the number of cost function evaluations used to reach the SchwefelValue by the generated program

avgCFESchwefel is the average value of the number of cost function evaluation reached by SOMA and DE in 100 times repeated simulations [48].

SchwefelValue is the value of reached extreme

If the number of cost functions were higher, the value is behaving according to equation (6.15).

$$\text{SchwefelValue} \quad |\text{CFESchwefel} - \text{avgCFESchwefel}| \tag{6.15}$$

In the case the algorithm was not successful in the Schwefel function but was successful in 1st De Jong function, the rules are similar as in the case of Schwefel function, as seen in equations (6.16) and (6.17).

$$|\text{CFEDeJong} - \text{avgCFEDeJong}| \,/\, \text{DeJongValue} \tag{6.16}$$

$$\text{DeJongValue} \quad (\ |\ \text{CFEDeJong} - \text{avgCFEDeJong}\ |\ + |\text{CFEDeJong} - \text{avgCFEDeJong}|) \tag{6.17}$$

where

CFEDeJong is the number of cost function evaluations used to reach the DeJongValue by the generated program:

avgCFEDeJong is the average value of the number of cost function evaluation reached by SOMA and DE in 100 times repeated simulations [48].

DeJongValue is value of the reached extreme.

In the case that the generated algorithm was not successful at all, the final equation is used (6.18).

$$\text{DeJongValue} \quad |\text{CFEDeJong}\ | \tag{6.18}$$

This is not the only way as to how to design a suitable cost function. This one differs from the previous one not only in including the number of cost function evaluations inside the CostFunction but also in the approach to the value of the extreme itself [48]. In previous cases, we used the original value of the extreme, but more suitable is to find the difference from the global extreme. Then we are close to zero value and it is more predictive.

6.2.3. Results

This section compares DE and SOMA with newly developed algorithms. All simulations here were performed with 2 dimensional benchmark functions. The following figures are histories of behaviour of the best individual in the

population - 100 times repeated for SOMA algorithms – 1st De Jong and Schwefel (Fig. 6.5). DE algorithms has its graphs of history in Fig. 6.3 a) for 1st De Jong and in Fig. 6.4 a) for Schwefel.

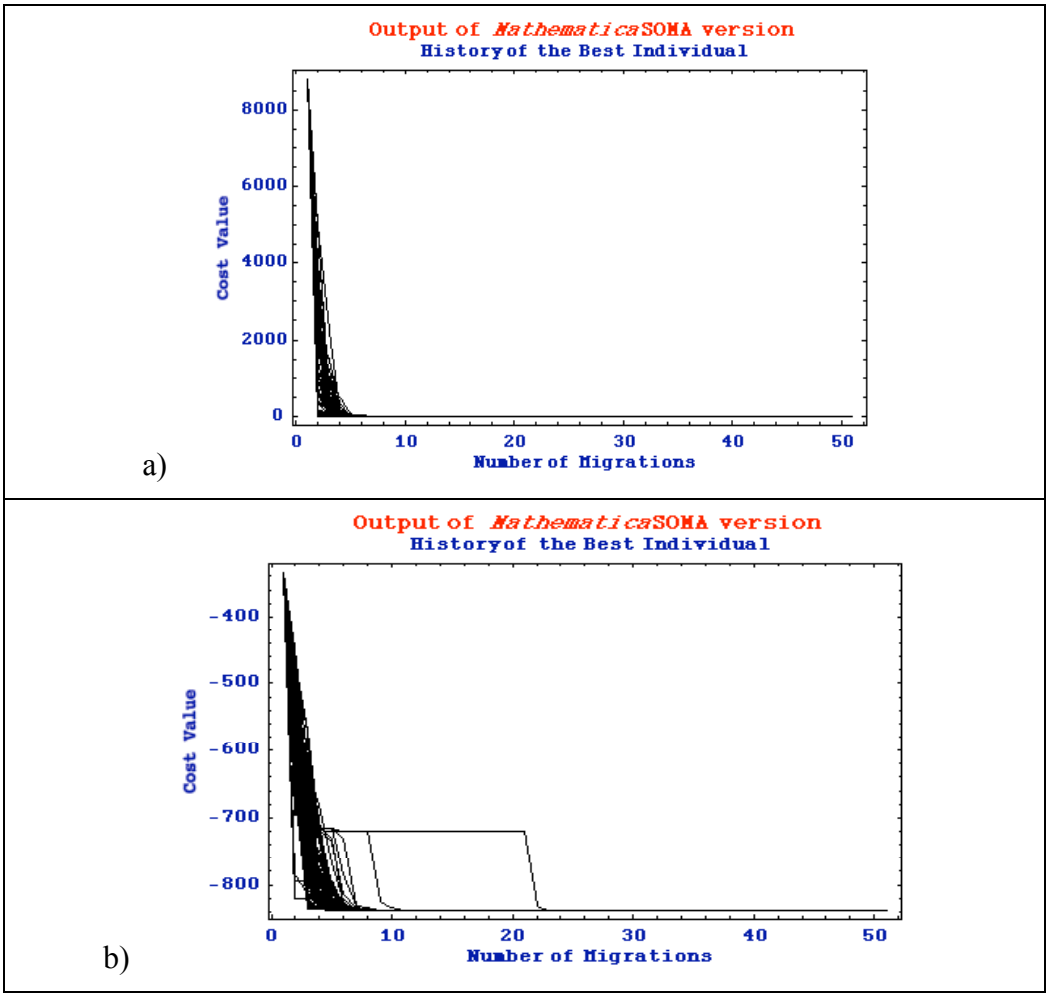


Fig. 6.5: 100times repeated for SOMA – a) 1st De Jong, b) Schwefel

The following table (Table 6.2) show values of extremes for 1st De Jong and Schwefel which were found by DE and SOMA for all 100 simulations.

Table 6.2: Values of extremes found by DE and SOMA

	Original DE		Original SOMA	
	1st De Jong	**Schwefel**	**1st De Jong**	**Schwefel**
Minimum	2.04492×10^{-8}	-837.966	2.39949×10^{-16}	-837.966
Maximum	6.61369×10^{-6}	-837.966	1.45227×10^{-14}	-837.966
Average	9.29224×10^{-7}	-837.966	3.6897×10^{-15}	-837.966

During our simulation we found successful and also non successful solutions.

As example, the following equations (6.19) - (6.22) belong to non successful solutions.

SelectDE (6.19)

SelectLeaderSOMA (6.20)

CrossDEBin(SelectDE) (6.21)

CompleteHC(SelectDE) (6.22)

The successful solution can be divided into two groups – which found subsolutions with requested diversity but the number of cost function evaluations were high and the final solution therefore was not so good (equations (6.23) and (6.24)). The second group contains solution which were successful in all conditions including original algorithms of SOMA and DE (equations (6.25) - (6.27)).

SOMAATOWithPRT(SOMAATORandWithPRT(SOMAATOWithoutPRT(MutateDERand1(SelectSOMALeader)))) (6.23)

SOMAATOWithPRT(MutateDEBest1(MutateDERand1(MutateDECurrentToBest(MutateDEBest1(MutateDECurrentToBest(SelectSOMARandLeader)))))) (6.24)

CrossDEBin(MutateRand1(SelectDE)) (6.25)

SOMAATORandWithPRT (SelectDE) (6.26)

MutateDEBest1(MutateDERand1(SelectSOMARandLeader)) (6.27)

Following Fig. 6.6 show graphs for 100times repeated simulations of algorithm with notation in (6.27).

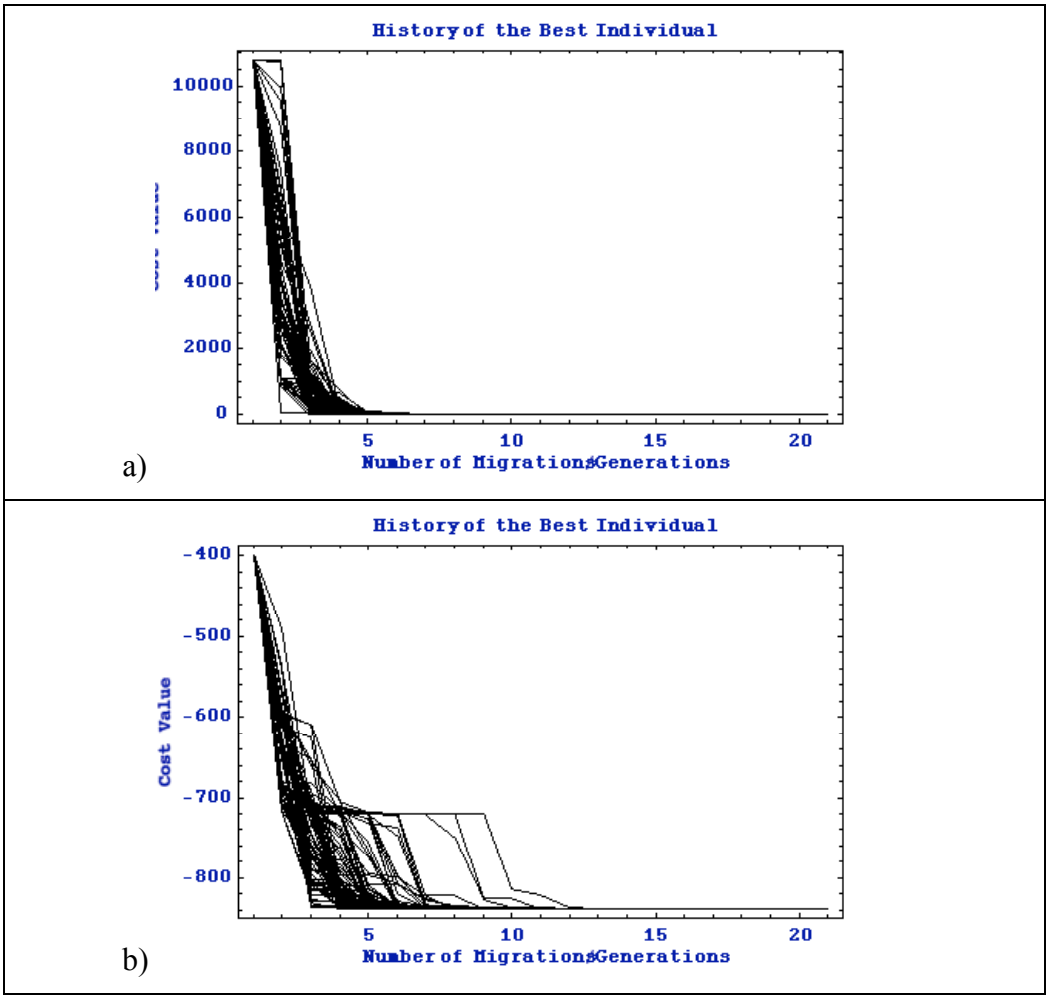


Fig. 6.6: 100times repeated for new algorithm - a) 1st De Jong, b) Schwefel

Table 6.3 shows values of extremes which were found by two new generated algorithms.

Table 6.3: Values of extremes for 1st De Jong and Schwefel found by new generated algorithms

	Generated algorithm (6.23)		Generated algorithm (6.24)	
	1st De Jong	**Schwefel**	**1st De Jong**	**Schwefel**
Minimum	5.86771×10^{-10}	-837.966	7.05752×10^{-10}	-837.966
Maximum	1.53905×10^{-4}	-800.053	5.90596×10^{-4}	-799.892
Average	9.06618×10^{-6}	-835.993	3.24827×10^{-5}	-835.871

The number of generations or migrations in new algorithms in the graph might be a little confusing. Number of cost function evaluations (CFE) in one loop for SOMA, DE and two new evolutionary algorithms are in equations (6.28) - (6.31). It means that 150 generations in DE means 3000 CFE if number of individuals is 20. Similar CFE (3109) in SOMA is for Migrations = 6. In new algorithms, 5 loops means 8282 and 3227 CFE for (6.23) and (6.24).

$$(\text{PopSize} - 1)\ \text{Migrations}\ (\text{PathLength} / \text{Step}) \tag{6.28}$$

$$\text{NP Generations} \tag{6.29}$$

$$\text{NP Generations}\ (3\ (\text{PathLength} / \text{Step}) + 1) \tag{6.30}$$

$$\text{NP Generations}\ ((\text{PathLength} / \text{Step}) + 5) \tag{6.31}$$

As can be seen, the generated programs were able to find minimum values, along with DE and SOMA. But not in all cases as Table 6.3 shows even if CFE is higher than in SOMA and DE. On the other hand the connection of several evolutionary operators show the promising approach, and its advantage which might occur in higher dimensional problems.

6.3. Higher dimensional problems

Usually in real life problems there is a need to find more optimal arguments than only two. Therefore we need higher dimensional problems to add into the costfunction for creating new optimization algorithms. In literature there appear usually optimizations of testing function in 20 or 100 dimension space.

We used 20 dimensional test function 1st De Jong and Schwefel, the same functions as in previous case. The settings in the costfunction for penalization were the same as above. Obviously only the average number of cost function evalutations for DE and SOMA in 100 repeated simulations had to be increased according to behaviour in 20 dimensional problems. 20 dimensional problems are more time consuming then only 2 dimension ones. This was the reason that it took several days to obtain first results. For finding algorithms we used iMac with 1.9 GHz PowerPC G5 processor, 512 MB RAM, Mac OS X version 10.4.8 and Mathematica 5.2.

Following tables (Table 6.4 - Table 6.7) shows settings for evolutionary operators as they were heuristically found as suitable in previous optimization tasks.

Table 6.4: Settings for SOMA operators

PathLengthIIII	3.
StepIIII	0.11
PRTIIII	.1
PopSizeIIII	60

Table 6.5: Settings for DE operators

CrIIII	0.8
FIIII	0.8
NPIIII	60

Table 6.6: Settings for HC

MaxIterIIII	500
StepHCIIII	2.3

Table 6.7: Settings for SA

TIIII	10 000
TminIIII	0.000 01
alphaIIII	0.91
MaxIterIIII	500
MaxIterTempIIII	100

The notation IIII is added to avoid confusion between settings for algorithms who will take care to create new evolutionary algorithms and settings for the simple operators.

In the case of synthesis, we cannot discuss about migrations or generations as this is unknown. We used marking iterations. Iterations were setup to 50. The process usually produce more complex structure and this is connected with one individual in the population. Although it is quite small number the cost function evaluations is then geometrically increased with usage of several operators in the line.

For the evolution with AP we used SOMA with following settings as given in Table 6.8.

Table 6.8: Settings for SOMA for AP

PathLength	3.
Step	0.22
PRT	.1
PopSize	20
Migrations	20

There were performed 100 cost functions evaluations until we obtained several results. Some successful, some not successful.

Examples of nonsuccessful algorithms according to conditions in cost function are given in following expressions (6.32) - (6.34):

CrossDEExp(SelectSOMALeader) (6.32)

CrossDEBin(CompleteHC(MutateDEBest1(CompleteSA(MutateDEBest2(
CrossDEBin(MutateDERand2(SOMAATOWithPRT(MutateDECurrentToBes
t(MutateDEBest2(SelectSOMARandLeader)))))))))) (6.33)

MutateDERand2(SelectSOMALeader) (6.34)

The first expression (6.32) is nonsuccessful because only crossover between the best one and the current individual is not enough in the requested cost functions evaluations. The second (6.33) expression is too complex and Hill Climbing and Simulated Annealing probably increase the number of cost function evaluations that the algorithms is not suitable for fast optimization. The last one is a surprise because it is the version of the Differential Evolution without crossing. The problem is in the number of iterations. 50 iterations in the case of unattached Differential Evolution is not enough. It needs more iterations (generations). The suitable settings of parameters is very hard optimization problem, thus the field of suitable settings is an open research area.

Into group of successful algorithms belongs (6.35) - (6.38):

SOMAATORandWithoutPRT(SOMAATORandWithPRT(SOMAATORand
WithPRT(MutateDECurrentToBest(SelectSOMALeader)))) (6.35)

SOMAATOWithPRT(SOMAATOWithPRT(SOMAATORandWithPRT(Cros
sDEBin(SOMAATOWithPRT(SelectSOMARandLeader))))) (6.36)

CrossDEBin(SOMAATOWithPRT(MutateDECurrentToBest(SelectSOMALe
ader))) (6.37)

SOMAATOWithPRT(SelectSOMALeader) (6.38)

Firstly, it is obvious that (6.38) can be written as SOMAATO; this notation is used later in tests.

The second view on this notations shows that Hill Climbing and Simulated Annealing were given out of these notations evolutionary. The local search increases the number of cost function evaluations. Presuambly, this is the reason why only Differential Evolution and SOMA remains in the generated algorithms.

6.3.1. Results

Algorithms in (6.35) - (6.38) were chosen for further simulations. We were interested in their behaviours in unimodal and multimodal benchmark functions in 2, 20 and 100 dimensional space.

This led to large number of simulations. We performed 16 benchmark functions for 4 algorithms, for 3 study cases (2, 20 and 100 Dim). And each case was 100 times repeated to produce graphs of history and convergence to the extreme. Total number was 19 200 of runs each algorithm where the number of cost function evaluations were from 70 000 to 300 000.

These simulations were taken on the XServe with 2x2 GHz Dual – Core Intel Xeon processors with 1 GB RAM, Mac OS X version 10.4.10 and gridMathematica 5.2. The whole machine contains 14 XServes, i.e. 56 processors together.

All graphs produced during the tests are shown in Appendix. Layout of the Appendix is as follows. Firstly the charts for 4 algorithms of 100 repeated simulations were recorded in sense of the best individual cost value per each iterations. The number on the x axis is given as the cost function evaluations (CFE) / number of iterations. CFE is there only for information because the most often comparing parameter between two evolutionary algorithms whereas the best individual was chosen in each iteration (generation, migration). The following lines are then indicated by the number of the algorithm (Algorithm 1 corresponds to (6.35), Algorithm 2 corresponds to (6.38), Algorithm 3 corresponds to (6.38) and Algorithm 4 corresponds to (6.38)). In each line, two graphs are shown. The one on the left side stands for

histograms of found extreme in each simulation. There is seen if the tests ended in the similar point or if the algorithm shows big diversity and then the irresponsibility.

The second chart shows the diversity in the final population for each performance. There is shown the minimum, maximum and average of found cost values. The diversity in population shows if the evolution still might make some progress or not. If the maximum is basically in minimum that the evolution is finished. The result is either the global extreme or local one, however this depends on the cost value itself.

In the case of 2 dimensional systems there are also figures of the functions itself in 3D charts. And on the right side there is contour plot with indicated points in the global extreme.

The last notice is concerned mainly to SOMAATO, which is the one of the "rediscovered" algorithms. All performed simulations for benchmark functions were done with the same settings as it was in the cost function for AP. No changes were made. It is obvious, that some problems would need a bit more sensible settings, mainly in 100 dimension space. The performed simulation might help in further work to change settings. Or there is open field for research of settings for algorithms itself – either by heuristic methods, or to use other evolutionary algorithms to tune their parameters.

6.3.2. Comments to behaviour of new algorithms itself

To show the success of the algorithms besides graphs; the results are also shown in the following tables (Table 6.9 - Table 6.24). For each case (2, 20 or 100 dimensional space) the minimal values were found in the final population. From this 100 values minimal, maximal and average value for each of 4 algorithms were counted.

Table 6.9: 1st De Jong's function

		Alg. 1	Alg. 2	Alg. 3	SOMAATO
2 DIM	Minimum	6.84058*10^-54	2.6887*10^-42	6.17178*10^-36	9.66931*10^-42
	Maximum	7.99154*10^-48	1.1667*10^-36	9.46824*10^-30	7.13651*10^-36
	Average	3.16793*10^-49	4.18871*10^-38	2.40994*10^-31	1.10491*10^-37
20 DIM	Minimum	1.82206*10^-6	0.0000621393	0.000624075	0.0000526722
	Maximum	0.000011683	0.000237911	0.00267865	0.000252083
	Average	4.84876*10^-6	0.000133213	0.00143861	0.00012626
100 DIM	Minimum	33.8511	2.47579	7.08902	2.48426
	Average	48.3372	4.32738	10.6768	4.25156
	Maximum	42.9626	3.37319	8.85669	3.16593

Table 6.10: 2nd De Jong's function

		Alg. 1	Alg. 2	Alg. 3	SOMAATO
2 DIM	Minimum	4.9499*10^-7	4.05763*10^-17	2.88213*10^-15	2.2027*10^-18
	Maximum	0.00632326	3.01539*10^-12	3.75052*10^-8	1.29338*10^-11
	Average	0.000471306	1.31109*10^-13	9.66692*10^-10	5.15334*10^-13
20 DIM	Minimum	12.4956	10.7737	10.1272	10.128
	Maximum	23.0267	16.7656	18.6797	16.7789
	Average	17.8876	14.965	15.978	14.4295
100 DIM	Minimum	1378.75	241.526	391.334	266.519
	Average	1861.5	472.805	614.45	457.46
	Maximum	1667.85	330.225	494.85	342.516

Table 6.11: 3rd De Jong's function

		Alg. 1	Alg. 2	Alg. 3	SOMAATO
2 DIM	Minimum	1.24955*10^-27	3.5426*10^-22	1.07207*10^-18	4.82185*10^-22
	Maximum	8.96953*10^-25	1.03514*10^-19	3.75781*10^-16	2.95776*10^-19
	Average	8.06571*10^-26	1.45729*10^-20	3.2283*10^-17	2.26685*10^-20
20 DIM	Minimum	0.0013398	0.00320009	0.0119396	0.00261852
	Maximum	0.00271373	0.00577972	0.0241444	0.00564708
	Average	0.00201386	0.00440167	0.0174949	0.004026
100 DIM	Minimum	15.6232	3.05624	5.73294	3.16925
	Average	18.8215	3.99875	7.25107	3.99408
	Maximum	17.4289	3.5365	6.39571	3.58527

Table 6.12: 4th De Jong's function

		Alg. 1	Alg. 2	Alg. 3	SOMAATO
2 DIM	Minimum	4.4845*10^-109	4.70679*10^-83	1.05796*10^-70	1.08294*10^-85
	Maximum	9.29256*10^-97	8.45534*10^-75	1.70096*10^-61	6.2068*10^-74
	Average	1.07945*10^-98	5.10371*10^-76	5.95297*10^-63	1.02549*10^-75
20 DIM	Minimum	1.51524*10^-14	8.08079*10^-10	6.5603*10^-8	6.27137*10^-10
	Maximum	2.54052*10^-12	1.5412*10^-8	1.19687*10^-6	1.14606*10^-8
	Average	5.94632*10^-13	5.8657*10^-9	2.94375*10^-7	4.56433*10^-9
100 DIM	Minimum	9.64008	0.224742	0.929832	0.197471
	Average	16.8637	0.643675	2.59482	0.59221
	Maximum	13.6604	0.354862	1.47562	0.334238

Table 6.13: Rastrigin's function

		Alg. 1	Alg. 2	Alg. 3	SOMAATO
2 DIM	Minimum	-400.	-400.	-400.	-400.
	Maximum	-400.	-400.	-400.	-400.
	Average	-400.	-400.	-400.	-400.
20 DIM	Minimum	-39093.5	-39984.5	-39714.1	-39969.7
	Maximum	-38047.3	-39390.	-38791.1	-39337.1
	Average	-38617.2	-39808.5	-39264.4	-39846.3
100 DIM	Minimum	-441350.	-764206.	-698468.	-761364.
	Average	-372123.	-695043.	-611882.	-700528.
	Maximum	-406800.	-731852.	-647676.	-728513.

Table 6.14: Schwefel's function

		Alg. 1	Alg. 2	Alg. 3	SOMAATO
2 DIM	Minimum	-837.966	-837.966	-837.966	-837.966
	Maximum	-837.966	-837.966	-837.966	-837.966
	Average	-837.966	-837.966	-837.966	-837.966
20 DIM	Minimum	-8354.52	-8379.46	-8376.91	-8379.39
	Maximum	-7953.3	-8378.27	-8365.21	-8376.88
	Average	-8168.74	-8379.06	-8373.98	-8378.77
100 DIM	Minimum	-24743.3	-33298.5	-30765.1	-33726.4
	Average	-21820.6	-30480.2	-27988.	-30663.5
	Maximum	-22875.9	-31852.3	-29258.7	-32152.6

Table 6.15: Griewangk's function

		Alg. 1	Alg. 2	Alg. 3	SOMAATO
2 DIM	Minimum	7.88258*10^-15	0.	0.	0.
	Maximum	0.00739606	0.	0.	0.
	Average	0.00040858	0.	0.	0.
20 DIM	Minimum	0.000104046	0.00144764	0.0235396	0.00214144
	Maximum	0.00565715	0.0303911	0.112688	0.0276332
	Average	0.00114136	0.00781872	0.0576792	0.00824393
100 DIM	Minimum	4.81464	1.24312	1.71041	1.25355
	Average	5.83553	1.486	2.06934	1.41581
	Maximum	5.28724	1.31854	1.85161	1.32937

Table 6.16: Sine Envelope Sine Wave function

		Alg. 1	Alg. 2	Alg. 3	SOMAATO
2 DIM	Minimum	-1.4915	-1.4915	-1.4915	-1.4915
	Maximum	-1.4915	-1.4915	-1.4915	-1.4915
	Average	-1.4915	-1.4915	-1.4915	-1.4915
20 DIM	Minimum	-27.4598	-28.0079	-27.7075	-27.9034
	Maximum	-26.2147	-27.3285	-26.5981	-26.9732
	Average	-26.8827	-27.666	-27.1445	-27.6299
100 DIM	Minimum	-86.6907	-109.765	-103.331	88.3073
	Average	-81.9419	-103.621	-97.0011	105.281
	Maximum	-84.1054	-106.933	-100.009	98.189

Table 6.17: Stretched V sine wave function - Ackley

		Alg. 1	Alg. 2	Alg. 3	SOMAATO
2 DIM	Minimum	0.000287748	3.32245*10^-9	1.75195*10^-8	1.54957*10^-9
	Maximum	0.0151492	2.29097*10^-7	8.09508*10^-6	6.20052*10^-7
	Average	0.00412367	3.98908*10^-8	9.96666*10^-7	4.74844*10^-8
20 DIM	Minimum	6.01572	1.65993	3.02977	1.45253
	Maximum	12.2744	2.72404	5.05216	2.74354
	Average	9.82864	2.13613	4.01447	2.08252
100 DIM	Minimum	213.189	89.708	112.611	87.4909
	Average	236.091	104.556	133.619	105.114
	Maximum	225.393	96.8063	125.024	96.469

Table 6.18: Ackley test function

		Alg. 1	Alg. 2	Alg. 3	SOMAATO
2 DIM	Minimum	-2.60065	-2.60065	-2.60065	-2.60065
	Maximum	-2.60065	-2.60065	-2.60065	-2.60065
	Average	-2.60065	-2.60065	-2.60065	-2.60065
20 DIM	Minimum	-30.8076	-31.0918	-30.9659	-31.0887
	Maximum	-29.5595	-30.0059	-29.2372	-30.284
	Average	-30.165	-30.9384	-30.2435	-30.9482
100 DIM	Minimum	294.8	-47.3369	37.991	-47.8128
	Average	350.958	-9.5177	96.2447	-8.01849
	Maximum	325.895	-29.0134	59.3382	-27.3333

Table 6.19: Ackley function

		Alg. 1	Alg. 2	Alg. 3	SOMAATO
2 DIM	Minimum	-4.4409*10^-16	-4.4409*10^-16	4.44089*10^-16	-4.4409*10^-16
	Maximum	-4.4409*10^-16	-4.4409*10^-16	3.15303*10^-14	-4.4409*10^-16
	Average	-4.4409*10^-16	-4.4409*10^-16	4.88498*10^-15	-4.4409*10^-16
20 DIM	Minimum	0.167231	0.237852	0.944661	0.220695
	Maximum	0.508953	0.561074	2.12588	0.483481
	Average	0.29824	0.362845	1.46952	0.359461
100 DIM	Minimum	1017.82	323.812	474.656	323.165
	Average	1107.55	396.045	570.201	384.988
	Maximum	1067.62	358.892	513.635	357.177

Table 6.20: Egg Holder function

		Alg. 1	Alg. 2	Alg. 3	SOMAATO
2 DIM	Minimum	-959.44	-959.641	-959.641	-959.641
	Maximum	-894.573	-956.915	-899.463	-956.897
	Average	-948.715	-957.569	-956.261	-957.571
20 DIM	Minimum	-12555.8	-14371.8	-13357.8	-13820.1
	Maximum	-10645.8	-11861.7	-10954.2	-11901.
	Average	-11388.7	-12904.6	-12095.3	-12832.4
100 DIM	Minimum	-33656.1	-31642.2	-30322.2	-32689.2
	Average	-28690.5	-27651.5	-26928.6	-27872.9
	Maximum	-31210.6	-29342.3	-28592.4	-29488.6

Table 6.21: Rana‘s function

		Alg. 1	Alg. 2	Alg. 3	SOMAATO
2 DIM	Minimum	-500.794	-500.802	-500.802	-500.802
	Maximum	-477.938	-480.662	-494.164	-489.806
	Average	-495.752	-499.691	-499.544	-499.354
20 DIM	Minimum	-7626.53	-7705.18	-7346.31	-7659.59
	Maximum	-6481.92	-6885.89	-6686.58	-6833.08
	Average	-6883.87	-7211.28	-6983.28	-7191.25
100 DIM	Minimum	-21918.9	-19703.8	-19309.7	-19732.9
	Average	-18861.4	-17464.9	-17167.6	-17538.3
	Maximum	-19801.7	-18483.	-18181.4	-18360.5

Table 6.22: Pathological function

		Alg. 1	Alg. 2	Alg. 3	SOMAATO
2 DIM	Minimum	7.52946*10^-9	1.44329*10^-14	1.93361*10^-11	0.
	Maximum	0.0000836098	0.0000705348	0.000251965	0.0000330035
	Average	8.09481*10^-6	2.30942*10^-6	7.72052*10^-6	1.40815*10^-6
20 DIM	Minimum	2.87102	3.2142	3.51303	3.25255
	Maximum	4.22087	4.43468	4.72801	4.33404
	Average	3.55241	3.93989	4.2	3.88867
100 DIM	Minimum	36.1381	37.6409	38.1266	37.9414
	Average	38.4161	39.5239	39.922	39.4065
	Maximum	37.5256	38.8196	39.1873	38.7657

Table 6.23: Michalewicz‘s function

		Alg. 1	Alg. 2	Alg. 3	SOMAATO
2 DIM	Minimum	-1.8013	-1.8013	-1.8013	-1.8013
	Maximum	-1.8013	-1.8013	-1.8013	-1.8013
	Average	-1.8013	-1.8013	-1.8013	-1.8013
20 DIM	Minimum	-19.8189	-19.8187	-19.8176	-19.8187
	Maximum	-19.8111	-19.8163	-19.796	-19.817
	Average	-19.8183	-19.8181	-19.8127	-19.8182
100 DIM	Minimum	-71.9513	-92.2251	-87.684	-91.3573
	Average	-67.2357	-88.3072	-83.8975	-88.3404
	Maximum	-69.0707	-89.8767	-85.7371	-90.0936

Table 6.24: Master’s cosine wave function

		Alg. 1	Alg. 2	Alg. 3	SOMAATO
2 DIM	Minimum	-1.	-1.	-1.	-1.
	Maximum	-0.961233	-1.	-1.	-1.
	Average	-0.998699	-1.	-1.	-1.
20 DIM	Minimum	-15.953	-18.1463	-16.8458	-17.9085
	Maximum	-14.0282	-14.9099	-14.3835	-14.7218
	Average	-14.7589	-16.3618	-15.4408	-16.2809
100 DIM	Minimum	-37.9154	-46.927	-40.2639	-45.1454
	Average	-31.6508	-36.7738	-33.7774	-37.8305
	Maximum	-34.3427	-40.5856	-36.7087	-40.9226

As the previous tables show it is hard to outline which algorithm is the best. The winners for each problem are divided according to dimensions and can be seen in Table 6.25.

Table 6.25: Winner for each benchmark function.

	Algorithm 1	Algorithm 2	Algorithm 3	SOMAATO
2 D	1, 3, 4, 5, 6, 8, 10, 11, 12, 15, 16	5, 6, 7, 8, 10, 11, 12, 13, 15, 16	5, 6, 7, 8, 10, 11, 12, 13, 15, 16	2, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16
20 D	1, 3, 4, 7, 11, 14, 15	5, 6, 8, 10, 12, 13, 16	2	9
100 D	12, 13, 14	1, 2, 3, 5, 7, 8, 15, 16,		4, 6, 9, 10, 11

The numbers are for each benchmark problem as follows: 1- 1st De Jong's function, 2 - 2nd De Jong's function, 3 - 3rd De Jong's function, 4 - 4th De Jong's function, 5 - Rastrigin's function, 6 - Schwefel's function, 7 - Griewangk's function, 8 - Sine Envelope Sine Wave function, 9 - Stretched V sine wave function - Ackley, 10 - Ackley test function, 11 - Ackley function, 12 - Egg Holder function, 13 - Rana's function, 14 - Pathological function, 15 - Michalewicz's function, 16 - Master's cosine wave function. If the same number appears in more cells on the same row it means that algorithms finished in the same cost value.

The algorithms compete between themselves. It is also almost impossible to say that one algorithm was the best one for the specific task. In some cases one algorithm was better in 20 dimensional problem but the second algorithm was better in 100 dimensions as is clearly visible from Table 6.25.

There is also seen that Algorithm 2 was the most successful in different tasks compared to Algorithm 3 which has not won in higher dimensions except one case.

The tables (Table 6.9 - Table 6.24 and Table 6.25) also showed that in 2 dimension version of benchmark functions there was no problem in finding the global extreme. In most case all four algorithms finished in the same cost

value. Some difficulties appeared in higher dimensions. In 20 dimenional space either Algorithm 1 or Algorithm 2 was the winner. In 100 dimension the most successful algorithm was Algorithm 2.

The differences in quality of solutions might be also given by different cost functions evaluations. These was given by the setting in the AP as we mentioned above. We were interested in behaviour according to the settings in AP and nothing was changed. The number of cost function evaluations for the performed cases are in the Table 6.26. As can be seen the Algorithm 2 has the biggest number compared to Algorithm 3 and SOMAATO which had only around a quarter. As described above, there is the probably connection between these results and cost function evaluations. Despite this fact, SOMAATO was able to win in several cases even with less CFE.

Table 6.26: Cost function evaluations for performed algorithms

Algorithm 1	241414.
Algorithm 2	321868.
Algorithm 3	80554.5
SOMAATO	80454.5

The cost function evaluations is the most discussed parameter because we need to have fast convergence with very high quality and therefore to have CFE as less as possible. We might also setup higher number of iterations because as can be seen even in graphs in appendix, most of 100 dimension simulations showed that more cost functions evaluations could lead to better results. The convergence was not finished.

6.3.3. Possible approach to giving a name to new algorithms

These were first simulations resulting in a small number of algorithms. Probably the system of giving names will be the sequence number. Later some of algorithms might get an additional name according the special properties. The future work suppose to prepare webpages with the description of the obtaining new optimization algorithms with evolutionary attributes where one of the sections will be also the list of generated algorithms with the sequence number and the notation of the algorithm.

CONCLUSIONS AND DISCUSSIONS

The main aim of the dissertation was to show that it is possible to create new optimization algorithms, probably of evolutionary character, by means of symbolic regression and tools of artificial intelligence.

The way to the main simulations went through several tasks which should prove that Analytic Programming which was chosen is able to work with symbolic regression and with linguistic terms as well as with numerical values.

Firstly, approximations of data were carried out with 4 study cases and 4 evolutionary algorithms as the optimizations tools which found the best fitting curve. To the group of selected cases for fitting data belongs: Quintic, Sextic problems as the representatives of polynomial functions of 5th and 6th order. The other two functions – Three Sine and Four Sine - contained also trigonometric functions Sin. However, all 4 functions were approximated by simple mathematical functions and operators like plus, minus, multiply, division and variable and constants. No trigonometric functions were inside. Although AP was able to find suitable curves which fitted data as well as possible, almost without any error. This was presented in 2003 [20].

The second task worked with operators for design of logical electronic circuits. We carried tasks from literature out as 3 to 6 – even parity and 3 to 6 – symmetry problems. In this task we proved that the AP is able to work with operators as AND, NAND, OR and inputs to design electronic circuit according requested behaviour which was give by truth tables. The article in a journal concerned to this topic was published in 2004 [22].

Third case was to show that we can use even functions for movement and synthesize a suitable trajectory for a robot. The robot has an aim to find a pick up all food on the defined grid. In this problem we even found better solutions then GP. At least, in the literature Koza give as an suitable example, which should be under requested number of steps, solution which does not fit it at all. We reached several times the requested number of steps. The paper was accepted in the international congress in 2005 [23] and also at the big conference concerned to evolutionary computation in 2006 [24].

The results of above described problems were comparable with known tool GP, but the performance was faster, probably because the AP can use arbitrary

evolutionary algorithms, not only genetic algorithms. This led to the choice of AP for further process of metaevolution – synthesis of evolutionary algorithms by means of symbolic regression.

There were performed other tasks like synthesis of neural networks [25], chaos synthesis [26], or solutions of differential equations [18] besides the presented work.

This work was concentrated into synthesis of optimization algorithms probably of evolutionary character. The first steps were published in Mendel 2006 conference [48] and ECMS 2007 [50] where the paper was awarded best paper.

Firstly, it was necessary to define operators of evolutionary algorithms. In first steps we have tried only to obtain one separated algorithm back into the whole. As described in section 6.1 we were successful because we obtained the differential evolution back. We also obtained another version of differential evolution as was found in later analysis. Performance of both algorithms are in the graphs, the new one had faster convergence in unimodal functions but slower in multimodal function.

In the next simulations, a cost function was adopted. To avoid problems with different orders of cost value of benchmark function which were in cost function in AP, counting the values was changed. Instead of direct number we used difference from global extreme. The cost values then were differences between achieved value and the real extreme value. Both cases then should reach the zero.

Consequently to that the penalization was added inside the cost function. The penalization supported less cost function evaluations then the average of SOMA and DE performance was found.

We found the original DE and SOMA algorithm as well as others. But that case was carried out only in 2 dimension problems. The real world, compared to that, needs more dimensional systems. Therefore the next step was carried out using 20 dimensional system of benchmark functions.

From these simulations, we obtained 4 algorithms which fulfil the request on cost function evaluations. Because the two benchmark functions used in AP were not enough to say that the algorithms are suitable, we carried out a big amount of simulations on 16 test functions, in 3 study cases – 2, 20 and 100 dimension spaces for all 4 algorithms. Each of this cases was 100 repeated.

The total amount of simulations were 19 200. The simulations took from minutes in the 2 dimension space to long hours – even 26 in 100 dimension space. All simulations were carried in Mathematica 5.2 environment.

The results are shown in tables and graphs in appendix. From that it is not possible to say if some algorithm is the winner. They compete not only in benchmark functions but even in the different dimensions in the frame of one of them. But on previous descriptions can be stated that all 4 algorithms were successful in 2 dimensions and all found the extreme. In higher dimensions there was a big competition, on the other hand all algorithms show the ability to optimize multimodal functions.

Settings for algorithms was and has been the biggest question which is hard to answer exactly. Therefore there will be open research field to tune the algorithm parameters and their comparison with others.

The total number of cost function evaluations during whole doctoral study in described cases was 4.011 milliards. The number consists of 4 millions of simulations for approximation data, 3.2 millions for Boolean parity and symmetry problems, 10.5 millions settings a trajectory for artificial ant, 150 millions during searching of new algorithms, 3840 millions during testing benchmark functions in three dimensions.

As the above described approach showed the AP is able to create new algorithms, however, this is not the only one of the point of view to synthesis of evolutionary algorithms by means of symbolic regression, i.e. to collect operators of known algorithms and try to create something new from them.

Another approach is to go to the lower level of creating the algorithm and this is to create some operator itself. This might bring also new robust optimization algorithms to the world. All might be added in future research in this field because optimization algorithms of high quality and fast convergence to the global extreme will be the most desired request in the field of optimization.

The main goals stated at the beginning of this thesis are fulfilled in previous chapters in the following way.

1. **To prove that Analytic Programming is able to do symbolic regression and to prove that Analytic Programming is also able to work with linguistic terms not only with numeric values or mathematical operators**

The experimental part showed in chapters dedicated to approximation of data (section 5.1), design of electronic circuits (section 5.2) and setting of a suitable trajectory for a robot (section 5.3) that AP is working and able to perform symbolic regression. *This goal has been reached.*

2. **- to try that a creation of a new optimization algorithms, probably of evolutionary character, is possible with AP**

   - to define several operators of evolutionary algorithms (like crossover, mutation, perturbation from SOMA, and others) which will be used as simple functions for AP

   - to define restrictions in Cost Function as inclusion of number of cost function evaluation into quality of solution

   - to try to create an evolutionary algorithm which will be at least as robust as some current algorithms are and further to compare its behaviour with current ones

The section 7 describes this points in detail. There is the progress from the first steps to the final results. *This could be considered the main result of the thesis.*

3. **to give comparisons between created and current evolutionary algorithms**

The last goal is fulfilled within 96 pages of graphs in appendix and 8 pages of tables in section 6.3.2 where results from 19 200 simulations are included. *These point seems to be fulfilled too.*

REFERENCES

[1] ZELINKA I., *Umělá inteligence v problémech globální optimalizace*, BEN, Praha, 2002, ISBN 80-7300-069-5

[2] KVASNIČKA V., POSPÍCHAL J., TIŇO P., *Evolučné algoritmy*, STU Bralislava, 2000, ISBN 80-227-1377-5

[3] DAWIS L., *Handbook of Genetic Algorithms*, International Thomson Computer Press, 1996, ISBN 1850328250

[4] PRICE K., STORN R. M., LAMPINEN J. A., *Differential Evolution : A Practical Approach to Global Optimization (Natural Computing Series),* Springer; 1 edition , 2005, ISBN: 3540209506

[5] ZELINKA I., *„SOMA – Self Organizing Migrating Algorithm“*, In: BABU B.V., ONWUBOLU G. (eds), *New Optimization Techniques in Engineering*, Springer-Verlag, 2004, ISBN 3-540-20167X

[6] EBERHART R., KENNEDY J., *Swarm Intelligence (The Morgan Kaufmann Series in Artificial Intelligence)*, Morgan Kaufmann, 2001, ISBN 1558605959

[7] DORIGO M., *Ant Colony Optimization and Swarm Intelligence*, Springer, 2006, ISBN 3540226729

[8] FARMER J.D., PACKARD N., PERELSON A., "The immune system, adaptation and machine learning", *Physica D*, vol. 2, pp. 187—204, 1986

[9] RUSSEL S. J., NORVIG P., *Artificial Intelligence: Modern Approach*, Prentice Hall, 1995, ISBN: 0131038052

[10] KIRKPATRICK S., GELATT C. D., VECCHI M. P., *Optimization by Simulated Annealing*, Science, 13 May 1983, Volume 220, Number 4598, p. 671 – 680

[11] BACK T., FOGEL D. B., MICHALEWICZ Z., *Handbook of evolutionary algorithms*, Oxford University Press, 1997, ISBN 0750303921

[12] KOZA J. R., *Genetic Programming,* MIT Press, 1998, ISBN 0-262-11189-6

[13] KOZA J. R. ET AL., *Genetic Programming III; Darwinian Invention and problem Solving*, Morgan Kaufmann Publisher, 1999, ISBN 1-55860-543-6

[14] www.genetic-programming.org

[15] O'NEILL M., RYAN C., *Grammatical Evolution. Evolutionary Automatic Programming in an Arbitrary Language*, Kluwer Academic Publishers, 2003, ISBN 1402074441

[16] www.grammatical-evolution.org

[17] O'SULLIVAN J., RYAN C., *An Investigation into the Use of Different Search Strategies with Grammatical Evolution,* Proceedings of the 5th European Conference on Genetic Programming, p.268 - 277, 2002, Springer-Verlag London, UK, ISBN:3-540-43378-3

[18] ZELINKA I., *Analytic Programming by Means of Soma Algorithm*. ICICIS'02, First International Conference on Intelligent Computing and Information Systems, Egypt, Cairo, 2002, ISBN 977-237-172-3

[19] ZELINKA I., *Analytic Programming by Means of Soma Algorithm.* Mendel '02, In: Proc. 8th International Conference on Soft Computing Mendel'02, Brno, Czech Republic, 2002, 93-101., ISBN 80-214-2135-5

[20] ZELINKA I., OPLATKOVA Z., *Analytic programming – Comparative Study.* CIRAS'03, The second International Conference on Computational Intelligence, Robotics, and Autonomous Systems, Singapore, 2003, ISSN 0219-6131

[21] ZELINKA I.,OPLATKOVA Z, NOLLE L., *Boolean Symmetry Function Synthesis by Means of Arbitrary Evolutionary Algorithms-Comparative Study*, ESM '2004, In: Proc. 18th European Simulation Multiconference, Magdeburg, Germany 2004

[22] ZELINKA I.,OPLATKOVA Z, NOLLE L., *Boolean Symmetry Function Synthesis by Means of Arbitrary Evolutionary Algorithms-Comparative Study*, International Journal of Simulation Systems, Science and Technology, Volume 6, Number 9, August 2005, pages 44 - 56, ISSN: 1473-8031, online http://ducati.doc.ntu.ac.uk/uksim/journal/Vol-6/No.9/cover.htm, ISSN: 1473-804x

[23] OPLATKOVA Z., *Optimal Trajectory of Robots Using Symbolic Regression*, In: CD-ROM of Proc. 56th International Astronautical Congress 2005, Fukuoka, Japan, 2005, paper nr. IAC-05-C1.4.07

[24] OPLATKOVA Z., ZELINKA I., *Investigation on Artificial Ant using Analytic Programming*, GECCO 2006, Seattle, Washington, USA, 8 – 12 July 2006, ISBN 1-59593-186-4

[25] ZELINKA I., VARACHA P., OPLATKOVA Z., *Evolutionary Synthesis of Neural Network*, Mendel 2006 – 12th International Conference on Softcomputing, Brno, Czech Republic, 31 May – 2 June 2006, pages 25 – 31, ISBN 80-214-3195-4

[26] ZELINKA I., CHEN G., CELIKOVSKY S., *Chaos Synthesis by Means of Symbolic Regression*, International Journal of Bifurcation and Chaos, in print

[27] JOHNSON COLIN G., *Artificial immune systems programming for symbolic regression*, In C. RYAN, T. SOULE, M. KEIJZER, E. TSANG, R. POLI, AND E. COSTA, editors, Genetic Programming: 6th European Conference, LNCS 2610, p. 345-353, 2003

[28] SALUSTOWICZ R. P., SCHMIDHUBER J., *Probabilistic Incremental Program Evolution*, Evolutionary Computation, vol. 5, nr. 2, 1997, pages 123 – 141, MIT Press, ISSN 1063-6560

[29] PATERSON N., *Genetic Programming with context sensitive grammars,* doctoral thesis, University of St. Andrews, 2003

[30] PATERSON N., LIVESEY M., *Distinguishing genotype and phenotype in genetic programming,* In KOZA, GOLDBERG, FOGEL & RIOLO, eds. Late Breaking Papers at GP 1996, MIT Press, 1996, ISBN 0-18-201-031-7

[31] JOHNSON C. G., *Artificial Immune System Programming for Symbolic Regression*, Lecture notes in Computer Sciences series, Springer, Volume 2610/2003, 2003, ISSN 0302-9743.

[32] FERREIRA C., *Gene Expression Programming: Mathematical Modeling by an Artificial Intelligence,* Springer, 2006, ISBN: 3540327967

[33] OLTEAN M., GROSAN C., *Evolving Evolutionary Algorithms using Multi Expression Programming*, The 7th European Conference on Artificial Life, September 14-17, 2003, Dortmund, Edited by W. Banzhaf (et al), LNAI 2801, pp. 651-658, Springer-Verlag, Berlin, 2003

[34] STINSTRA E., RENNEN G., TEEUWEN G., *Meta-modelling by symbolic regression and Pareto Simulated Annealing,* Tilburg University, Netherlands, nr. 2006-15, ISSN 0924-7815

[35] DAVIDSON J.W., SAVIC D.A., WALTERS G.A., *Symbolic and numerical regression: experiments and applications*, Informatics and Computer Science – An international Journal, Vol. 150, Elsevier, USA, 2003, pages 95 – 117, ISSN:0020-0255

[36] Wikipedia – free encyclopaedia, en.wikipedia.org

[37] HOLLAND J. H., *Genetic Algorithms*, Scientific American, July 1992, p. 44 – 50

[38] BEASLEY D., BULL D. R., MARTIN R. R., *An Overview of Genetic Algorithms, Part 1, Fundamentals,* University Computing, 1993, 15(2), p. 58 – 69

[39] BEASLEY D., BULL D. R., MARTIN R. R., *An Overview of Genetic Algorithms, Part 2,* Research Topics, University Computing, 1993, 15(4), p. 170-181

[40] RATLE A., SEBAG M., *Genetic Programming and Domain Knowledge: Beyond the Limitations of Grammar-Guided Machine Discovery, Parallel Problem Solving from Nature* - PPSN VI, 6th International Conference, Paris, France, Proceedings. Lecture Notes in Computer Science 1917 Springer 2000, pages 211-220, ISBN 3-540-41056-2

[41] KEIJZER M., *Improving Symbolic Regression with Interval Arithmetic and Linear Scaling,* Lectures in Computer Science, Volume 2610, EuroGP, Springer, 2003, pages 70-82, ISSN: 0302-9743

[42] KEIJZER M*., Genetic Programming for Symbolic Regression, tutorial*, Gecco 2006, Seattle, Washington, 2006, ISSN: 0302-9743

[43] REKTORYS K., *Variational methods in Engineering Problems and Problems of Mathematical Physics*, Czech edition, 1999, ISBN 80-200-0714-8

[44] LAMPINEN J., ZELINKA I., *New Ideas in Optimization – Mechanical Engineering Design Optimization by Differential Devolution*, Volume 1. London: McGraw-hill, 1999, 20 p., ISBN 007-709506-5

[45] OŠMERA P., *Evolution of Complexity*, In: Li Z., Halang W. A., Chen G.: Integration of Fuzzy Logic and Chaos Theory, Springer-Verlag, 2006, ISBN: 3-540-26899-5, pages 527 – 578

[46] MAŘÍK V. ET AL., *Artificial Intelligence IV*., 2003, Academia, Praha, Czech edition, ISBN 80-200-1044-0

[47] KOZA J. R., KEANE M. A, STREETER M. J., *Evolving Inventions*, Scientific American, February 2003, p. 40-47, ISSN 0036-8733, (online www.sciam.com)

[48] OPLATKOVA Z., ZELINKA I., *Creating evolutionary algorithms by means of analytic programming – preliminary study*, Mendel 2006, Brno, Czech Republic, 31 May – 2 June 2006, pages 19 – 24, ISBN 80-214-3195-4

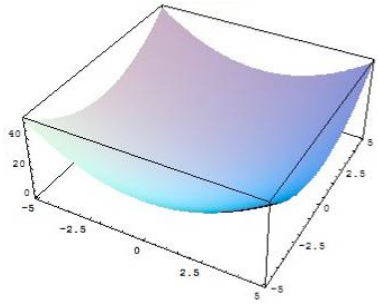
[49] BABU, B.V., *"Evolutionary Computation - At a Glance",* NEXUS, Annual Magazine of Engineering Technology Association, BITS, Pilani, pp. 3-7, 2001

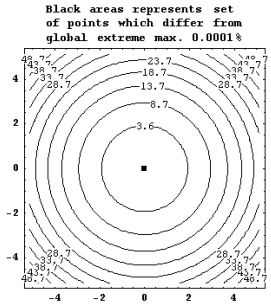
[50] OPLATKOVA Z., ZELINKA I., *Creating evolutionary algorithms by means of analytic programming – design of new cost function*, ECMS 2007, Praha, 3 – 6 June 2007, pages 271 – 276, ISBN 978-0-9553018-2-7

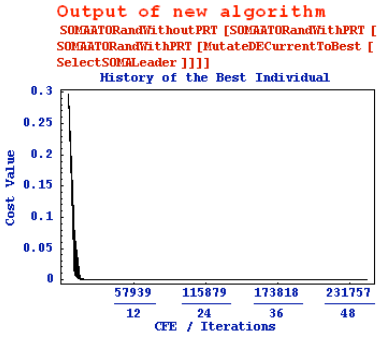
7. APPENDIX - TEST FUNCTIONS

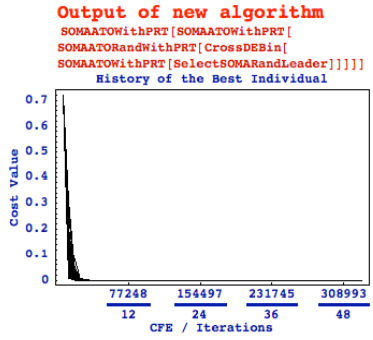
7.1. Sphere model, 1st De Jong's function – 2D

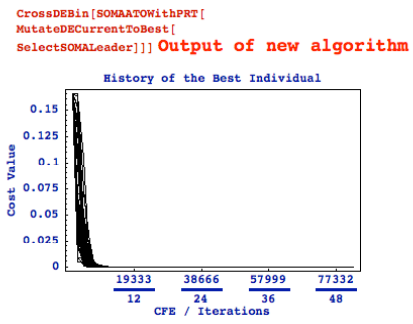
$$f_1(x) = \sum_{i=1}^{n} x_i^2$$

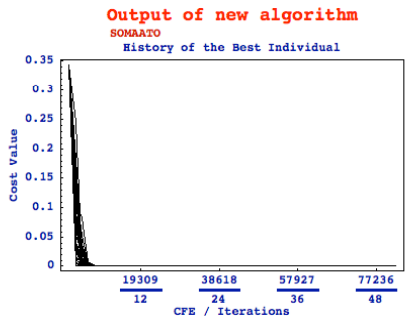




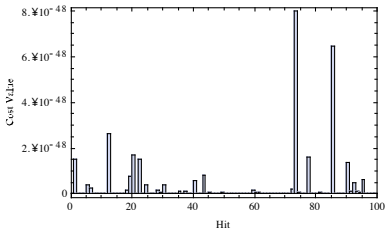


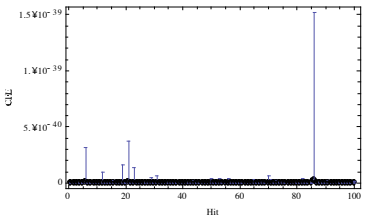




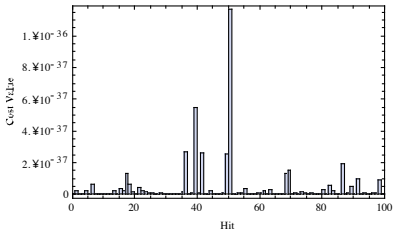


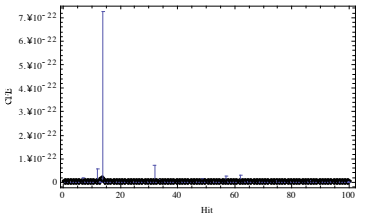
Algorithm 1



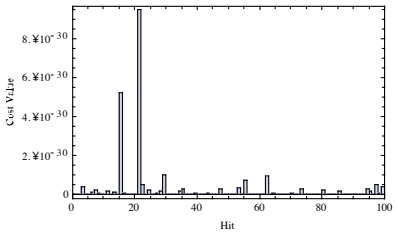


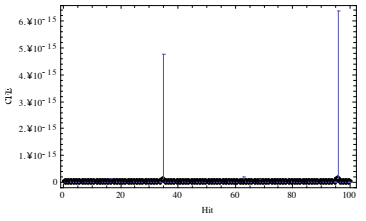
Algorithm 2



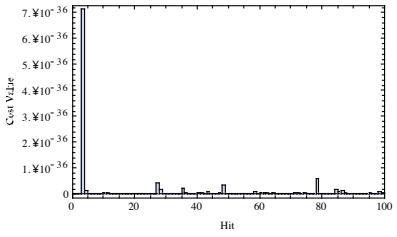


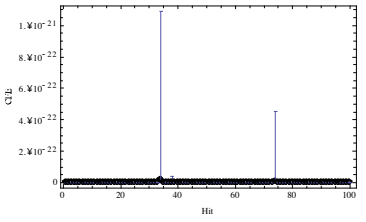
Algorithm 3



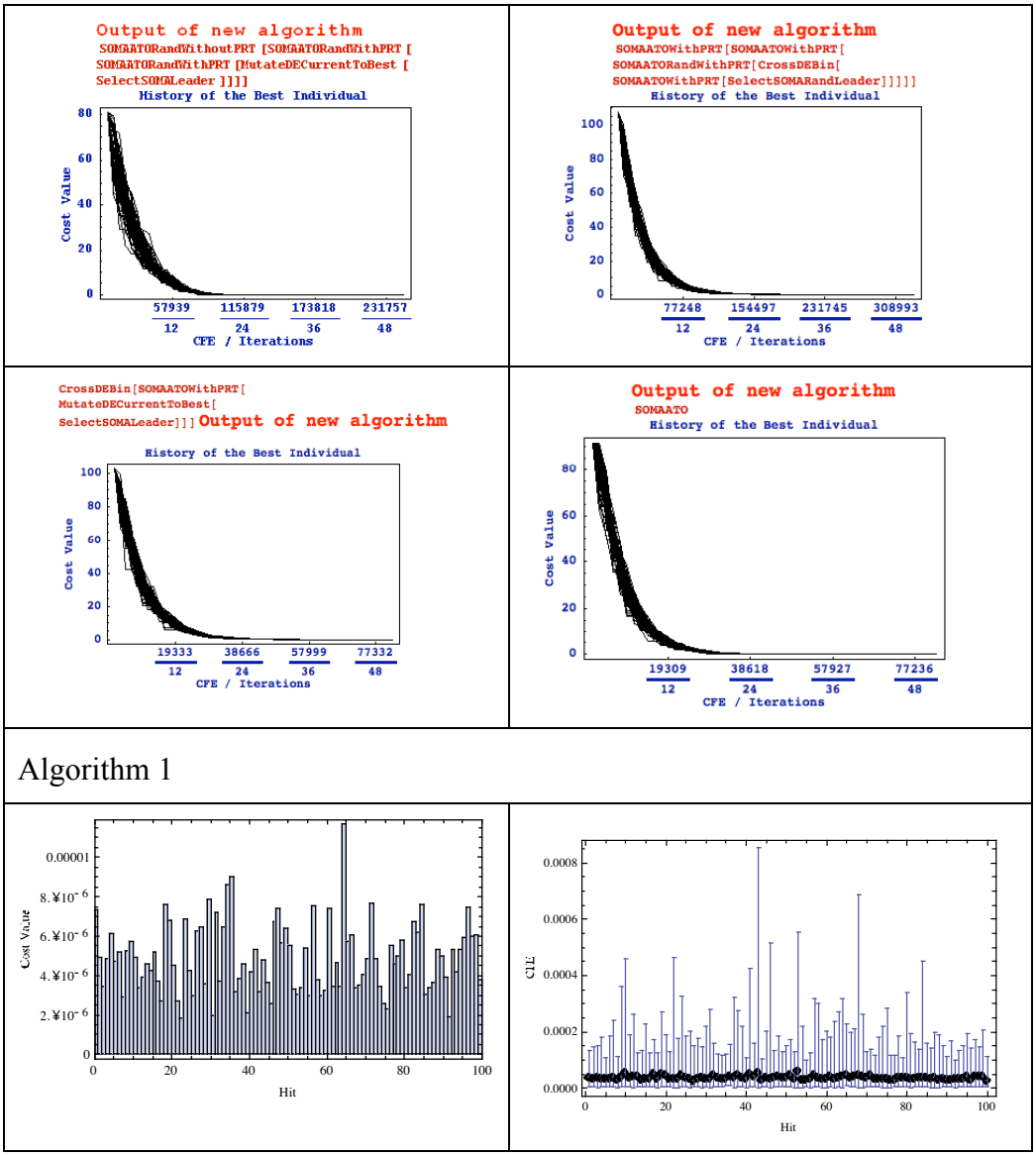


SOMAATO



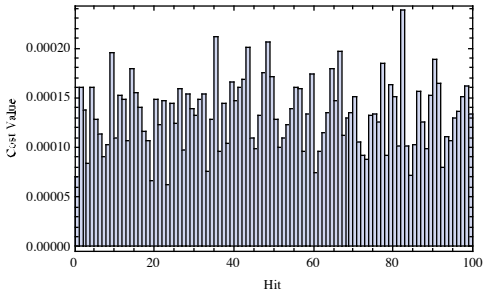


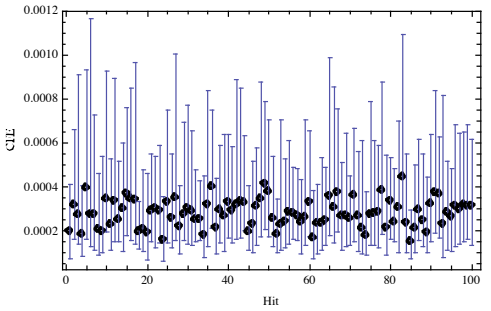
7.2. Sphere model, 1st De Jong's function – 20D



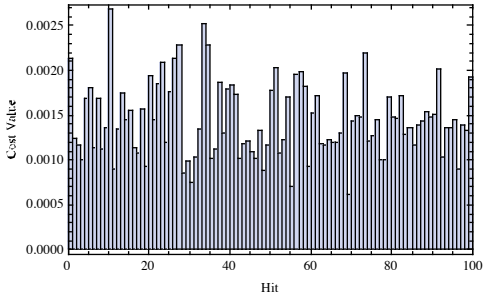
Algorithm 1

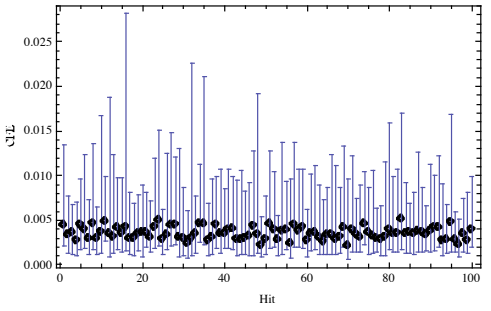
Algorithm 2



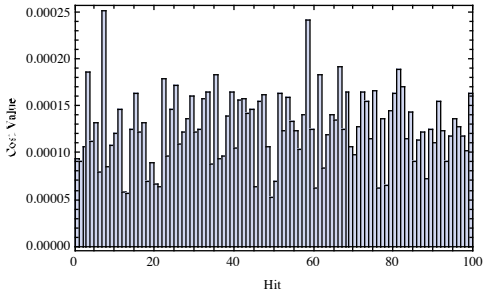


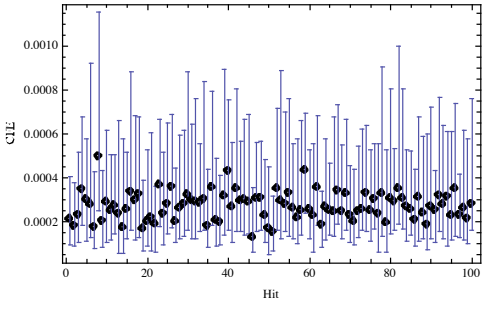
Algorithm 3



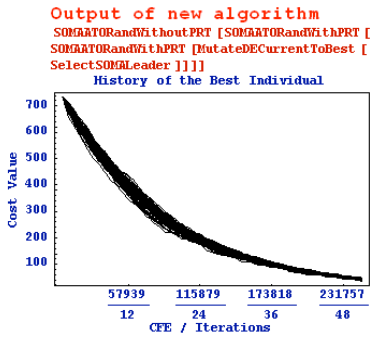


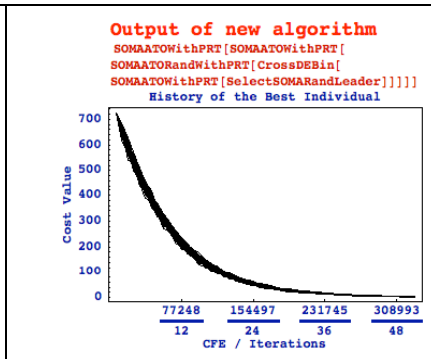
SOMAATO

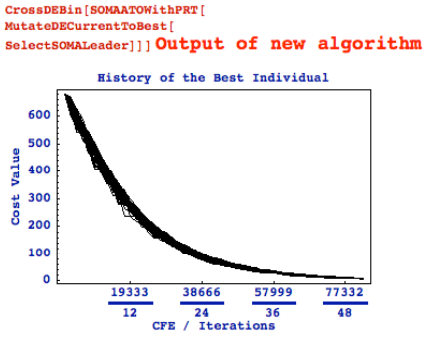


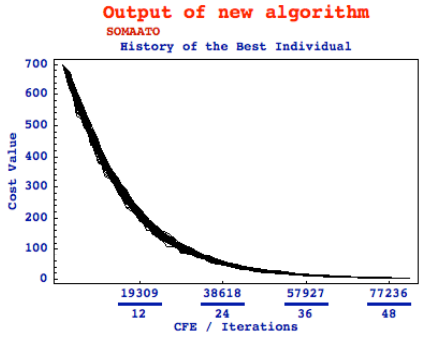


7.3. Sphere model, 1st De Jong's function – 100D

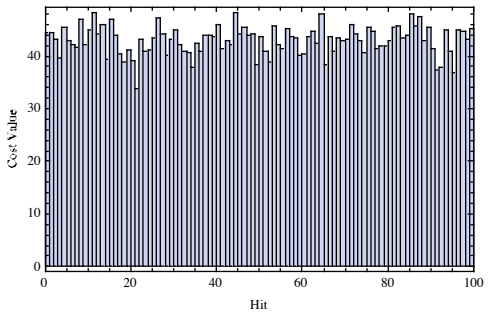


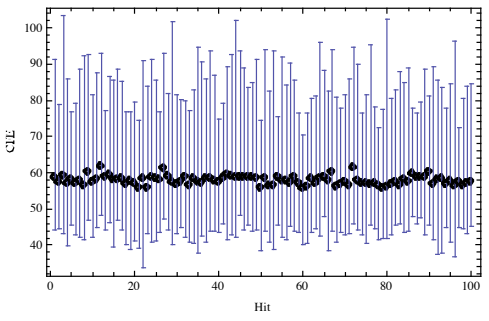




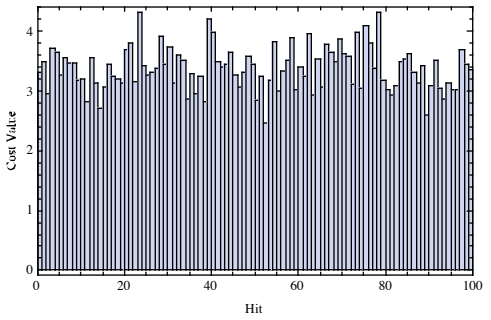


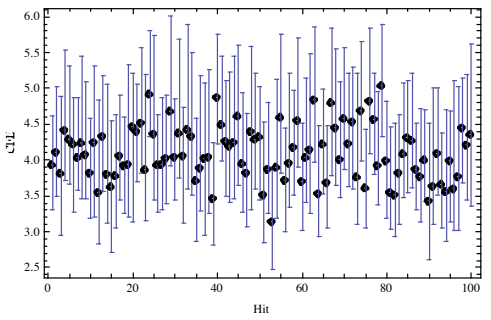
Algorithm 1



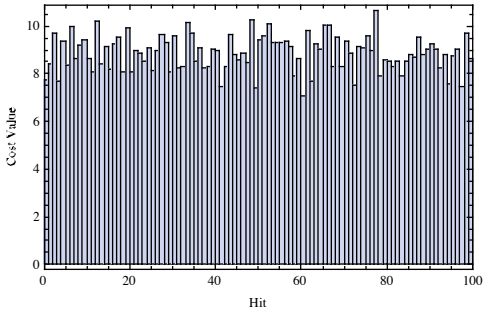


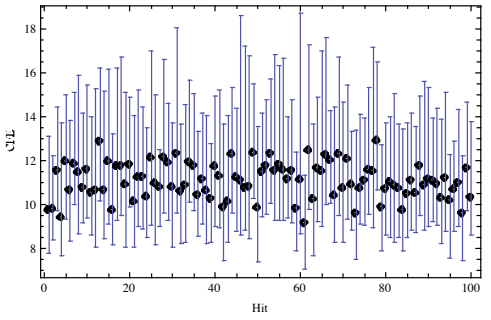
Algorithm 2



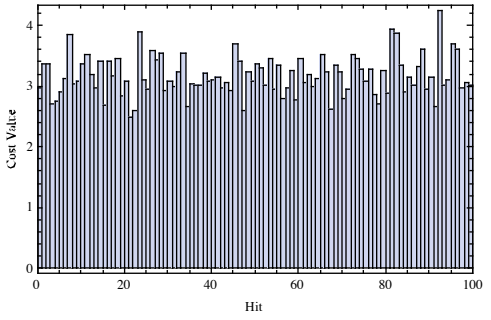


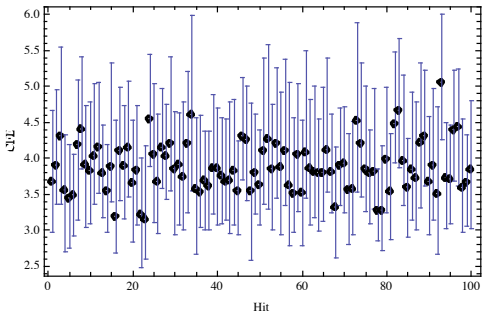
Algorithm 3





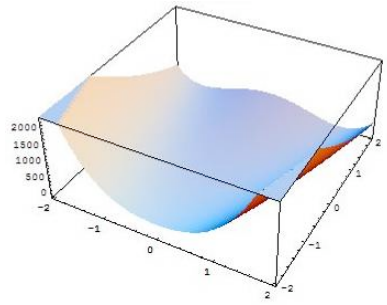
SOMAATO

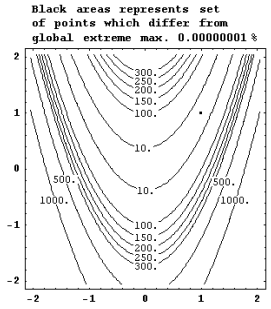


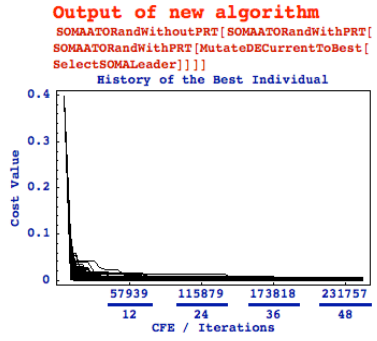


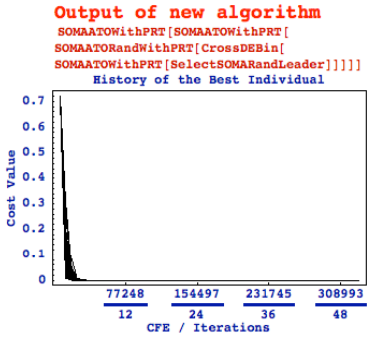
7.4. Rosenbrock‘s saddle, 2nd De Jong‘s function – 2D

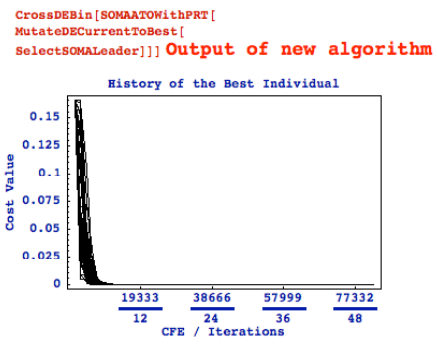
$$f_2(x) = \sum_{i=1}^{n-1} 100 \cdot \left(x_{i+1} + x_i^{\,2}\right)^2 + \left(1 - x_i\right)^2$$

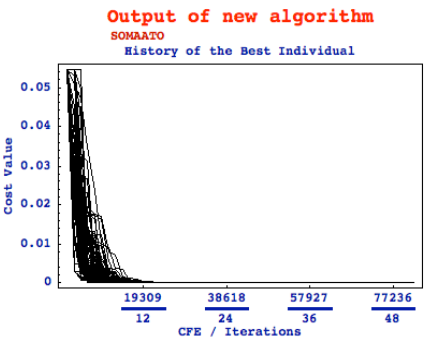




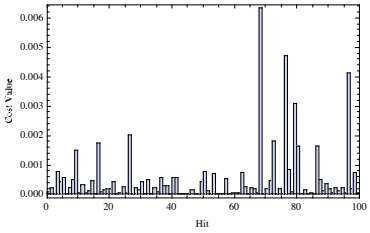


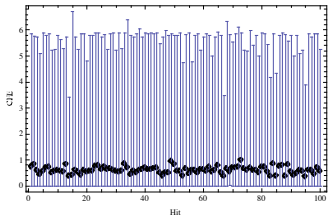




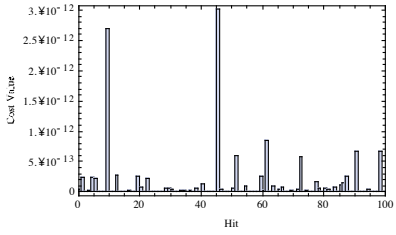


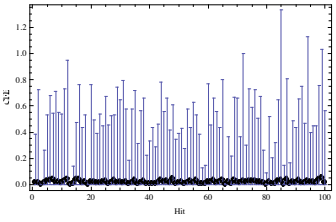
Algorithm 1



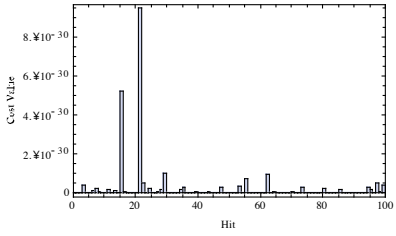


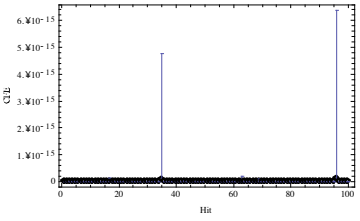
Algorithm 2



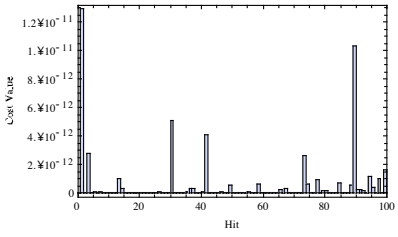


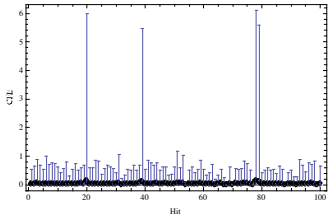
Algorithm 3



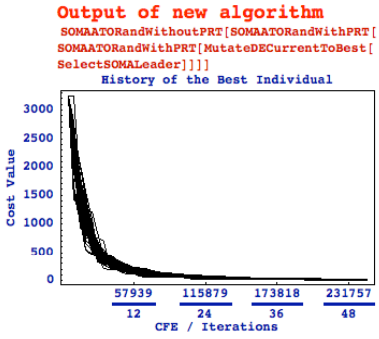


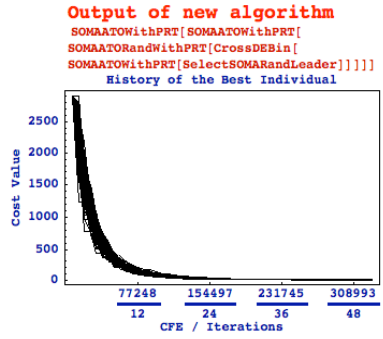
SOMAATO

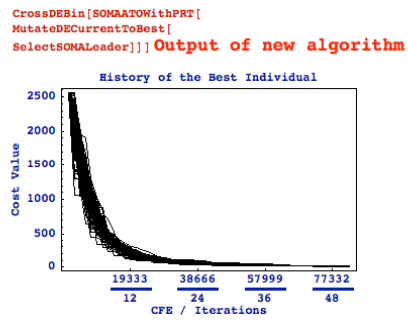


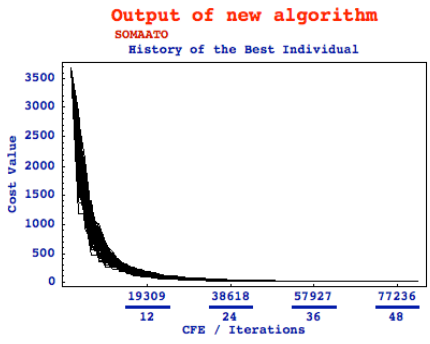


7.5. Rosenbrock‘s saddle, 2nd De Jong‘s function – 20D

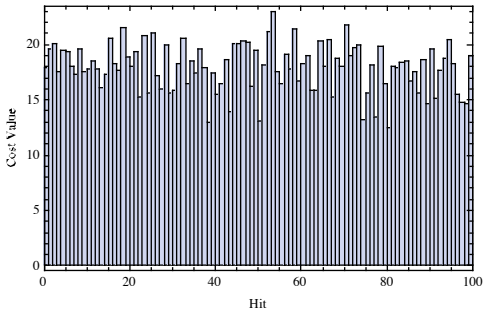


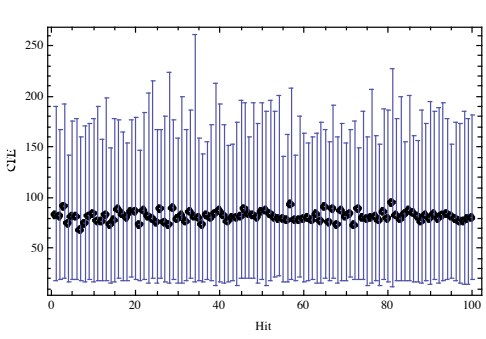




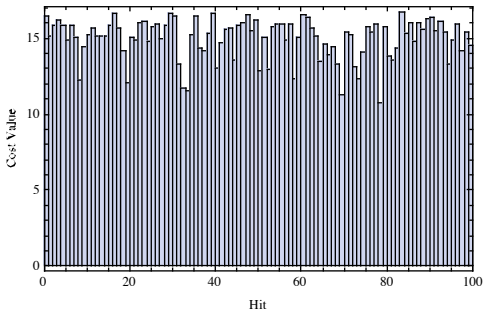


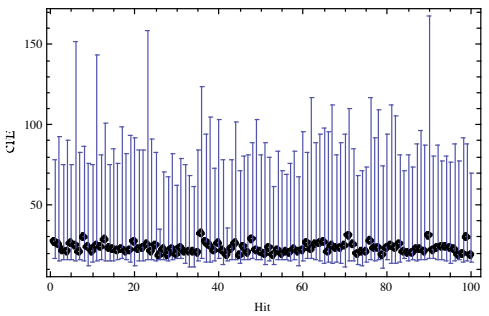
Algorithm 1



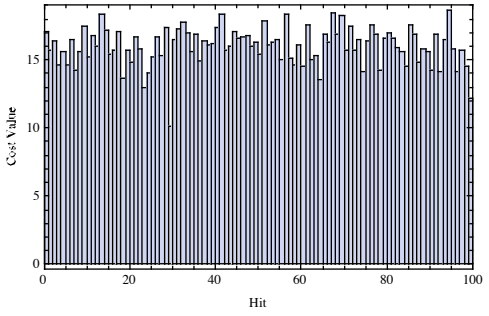


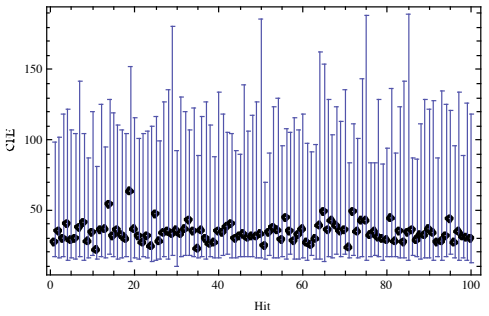
Algorithm 2



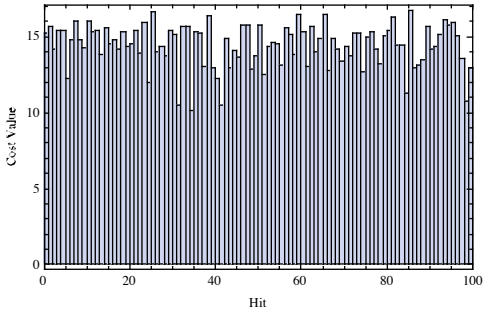


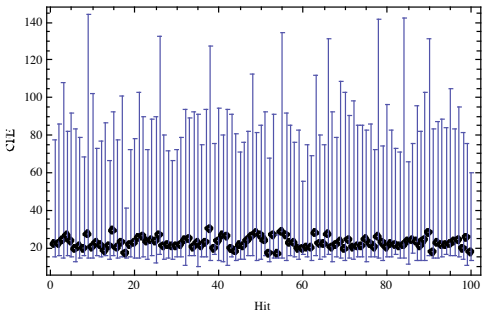
Algorithm 3



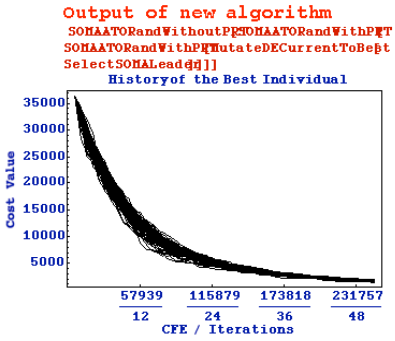


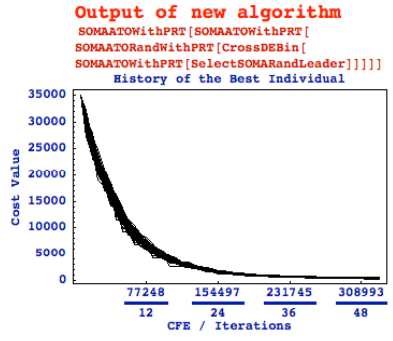
SOMAATO

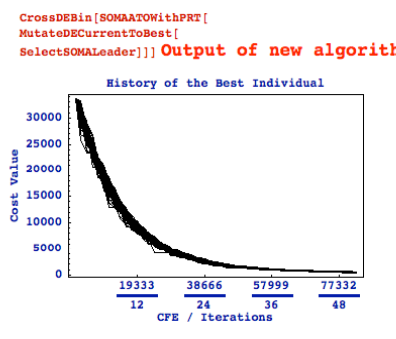


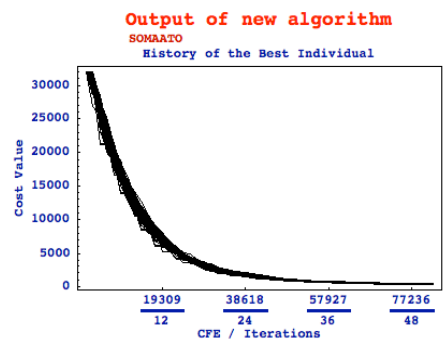


7.6. Rosenbrock‘s saddle, 2nd De Jong‘s function – 100D

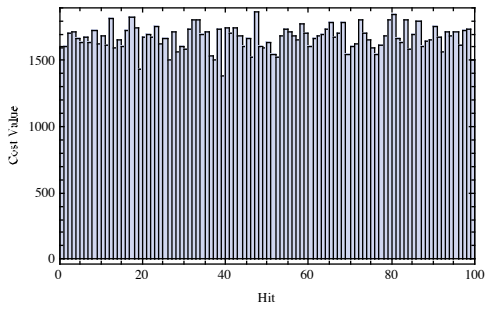


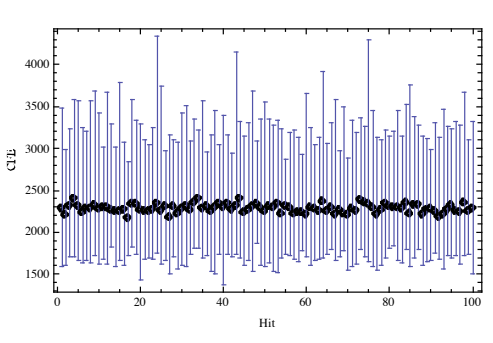




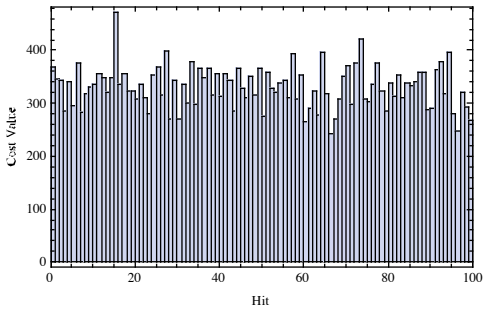


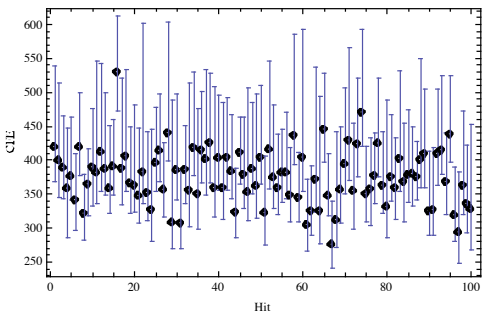
Algorithm 1



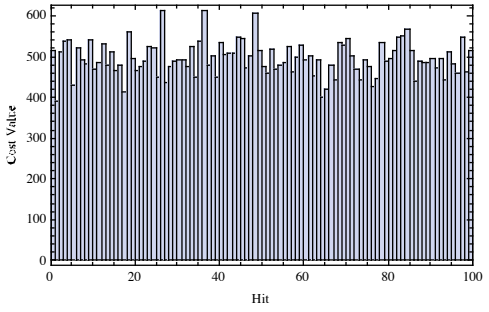


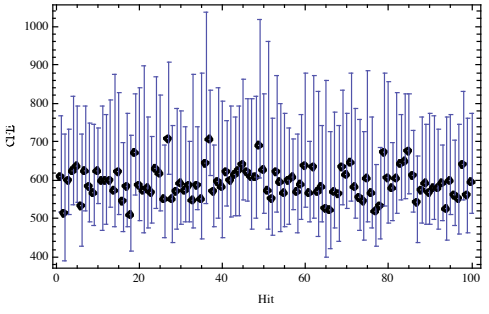
Algorithm 2



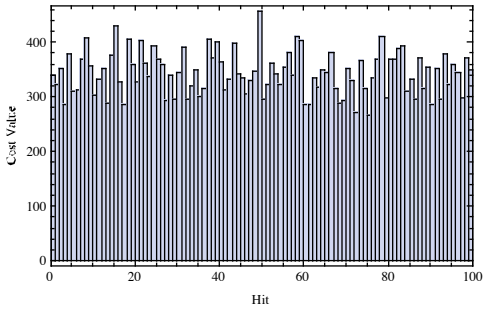


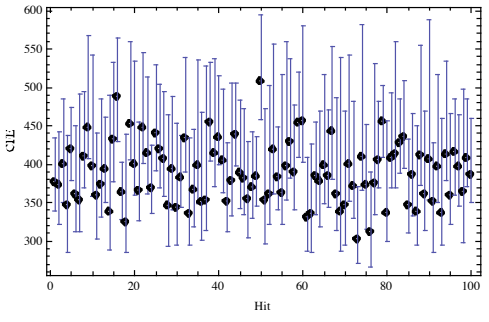
Algorithm 3





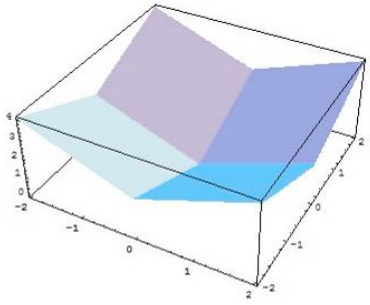
SOMAATO

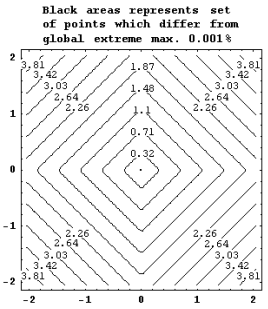


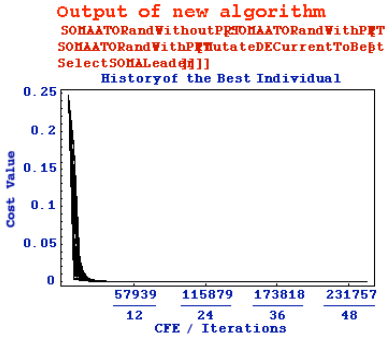


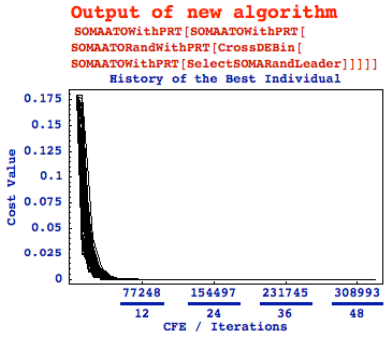
7.7. 3rd De Jong's function – 2D

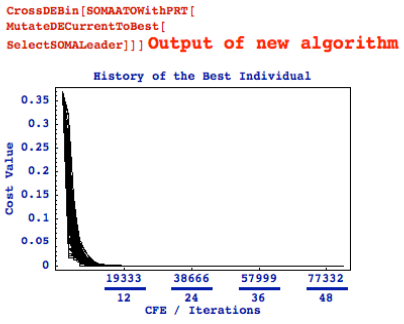
$$f(x) = \sum_{i=1}^{Dim} |x_i|$$

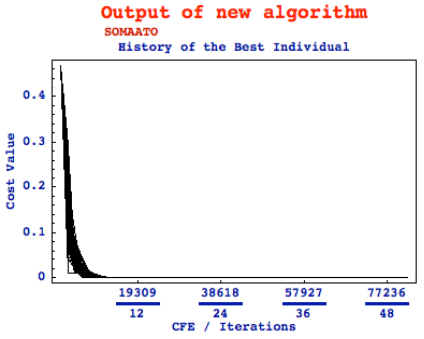




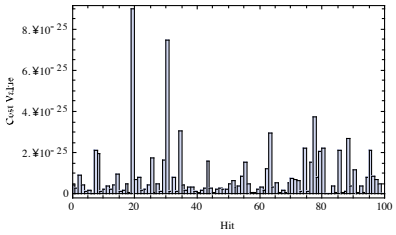


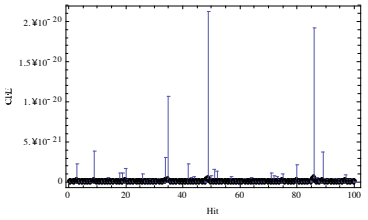




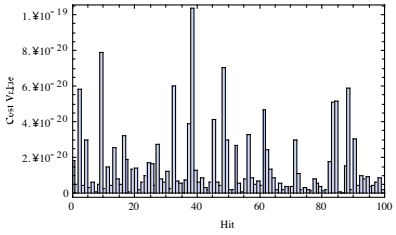


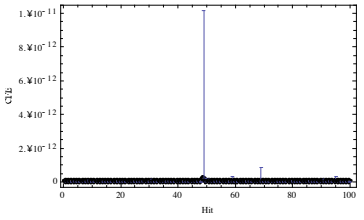
Algorithm 1



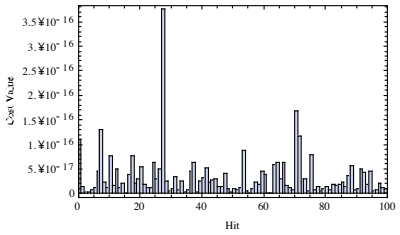


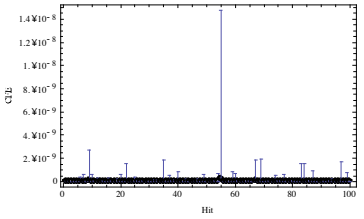
Algorithm 2



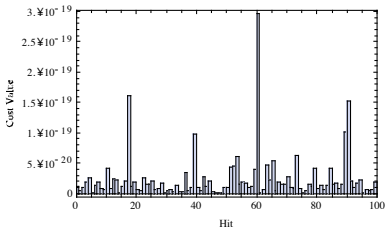


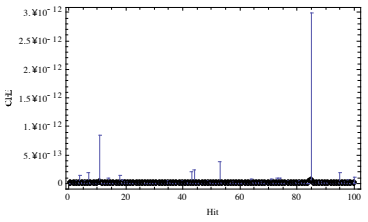
Algorithm 3



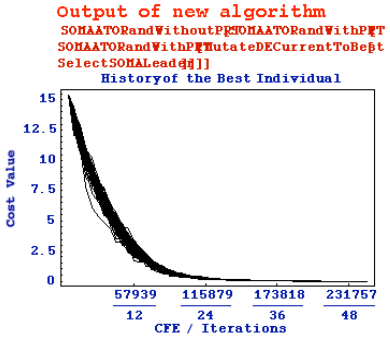


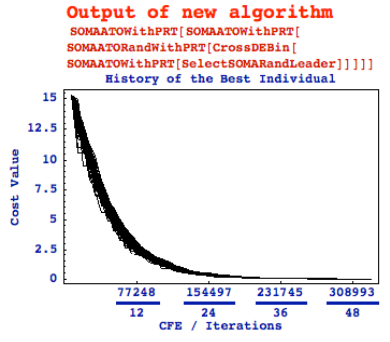
SOMAATO

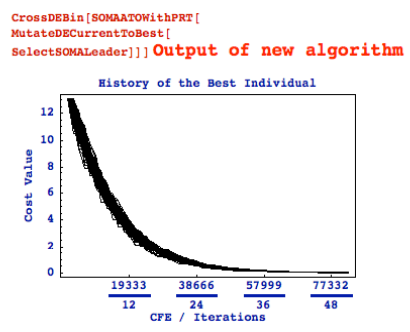


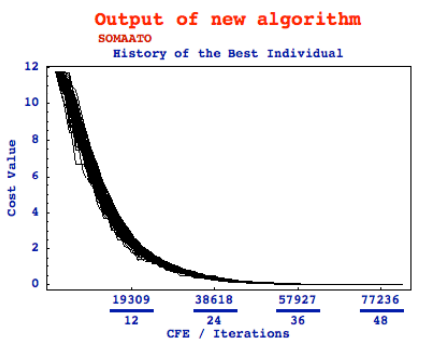


7.8. 3rd De Jong's function – 20D

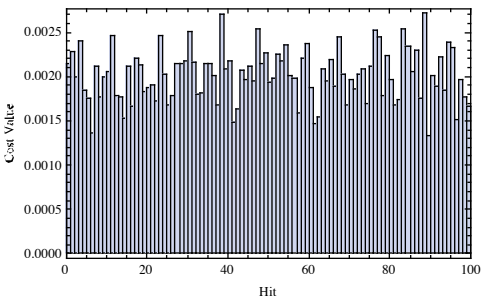


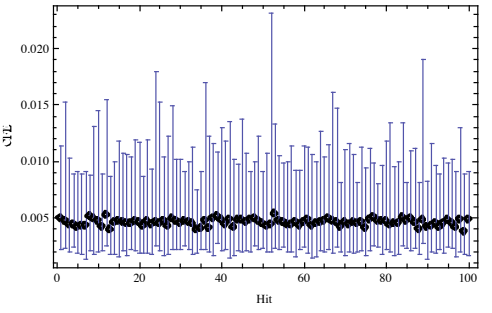




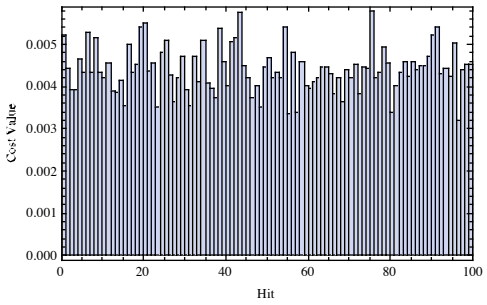


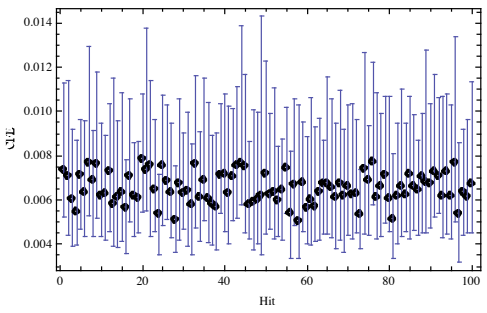
Algorithm 1



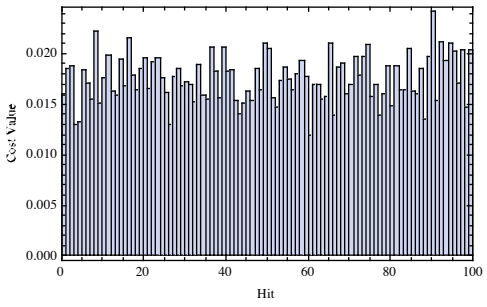


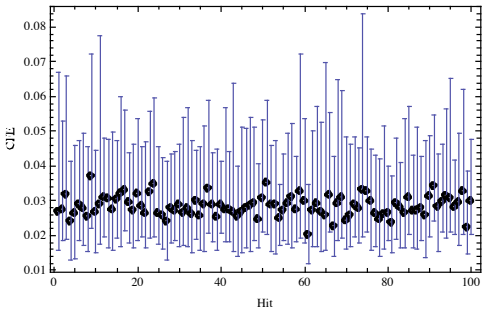
Algorithm 2



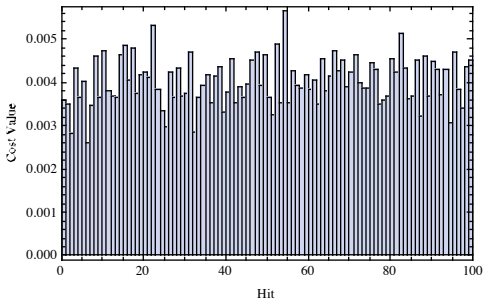


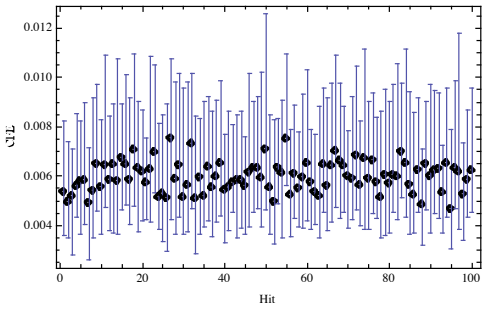
Algorithm 3



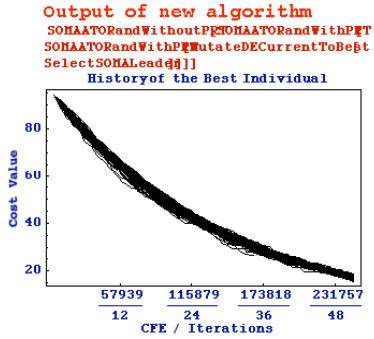


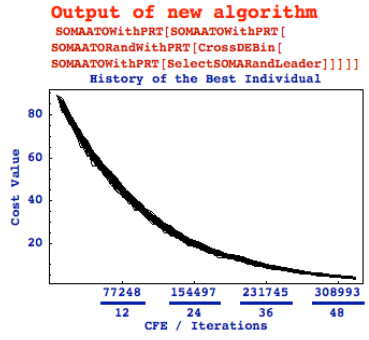
SOMAATO

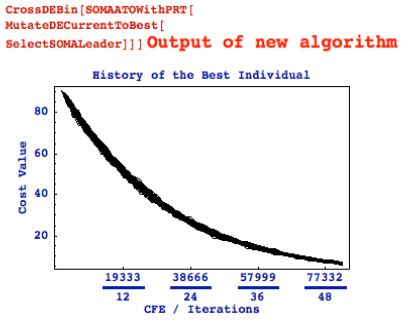


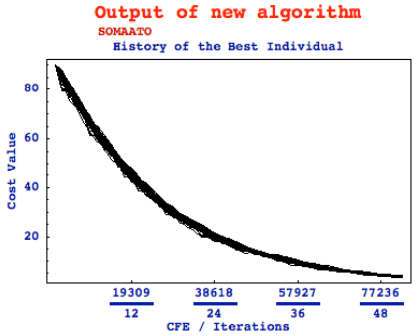


7.9. 3rd De Jong's function – 100D

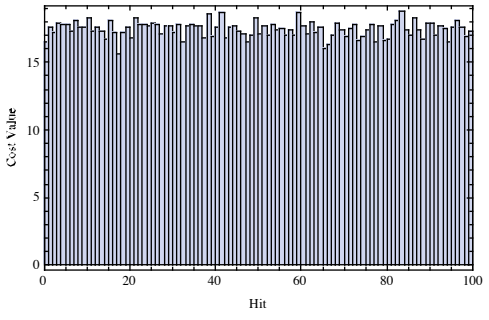


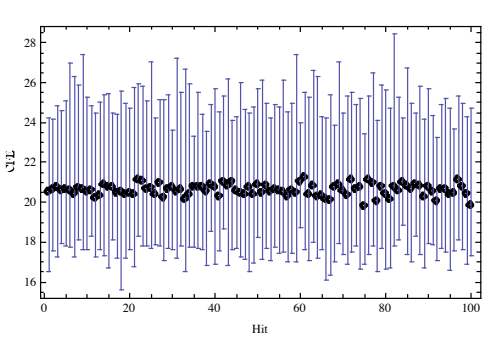




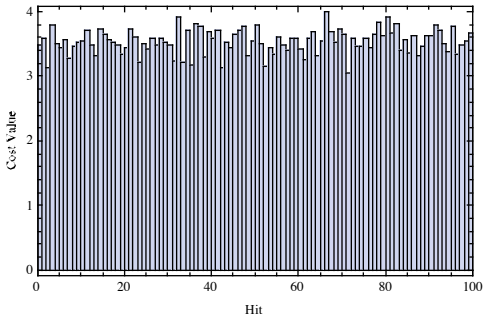


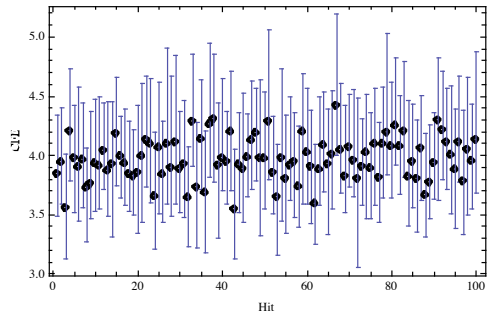
Algorithm 1



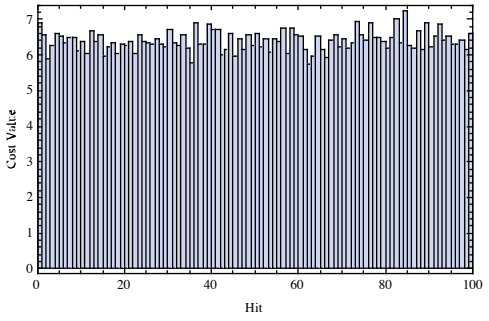


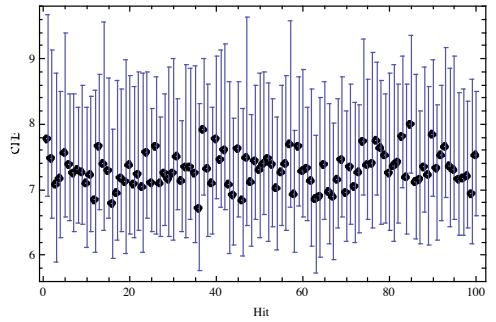
Algorithm 2



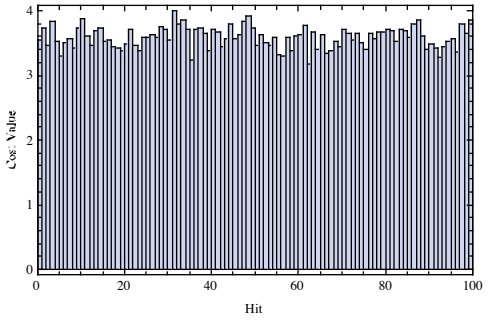


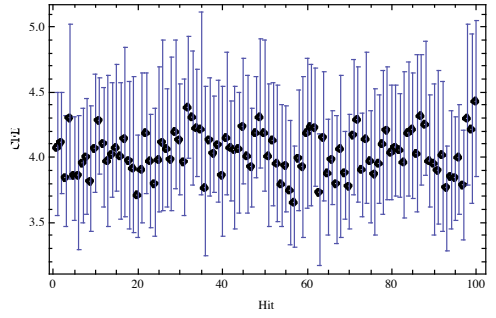
Algorithm 3





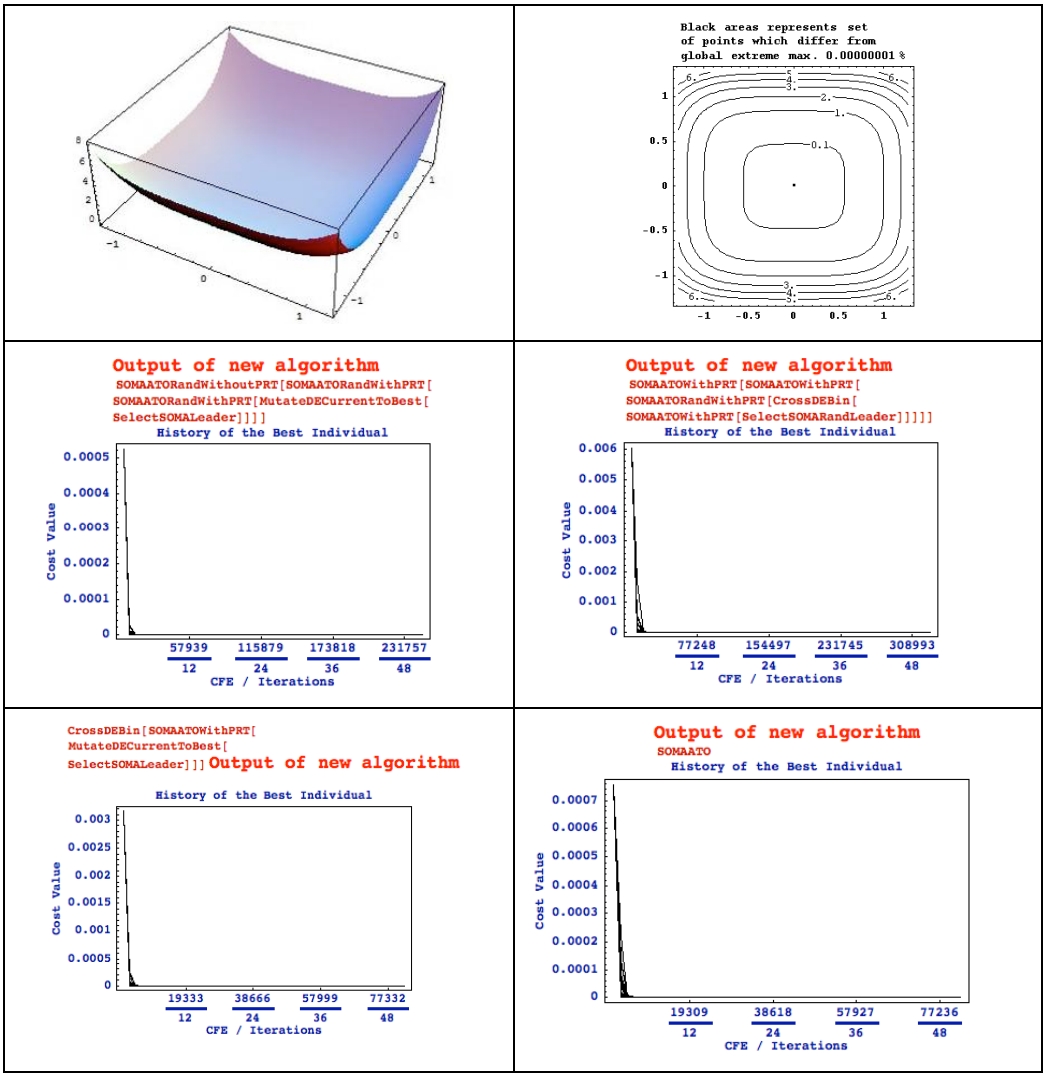
SOMAATO



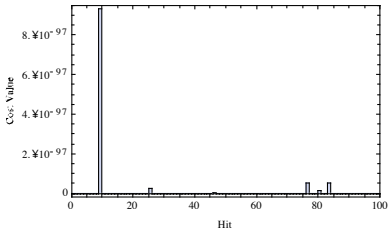


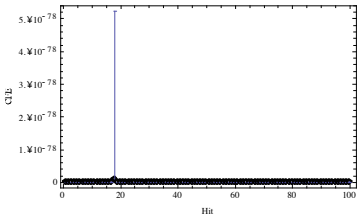
7.10. 4th De Jong's function – 2D

$$f(x) = \sum_{i=1}^{Dim} i x_i^4$$

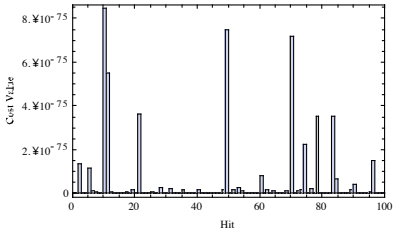


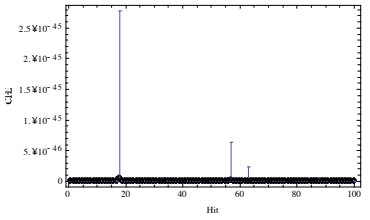
Algorithm 1



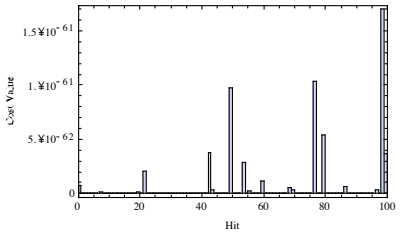


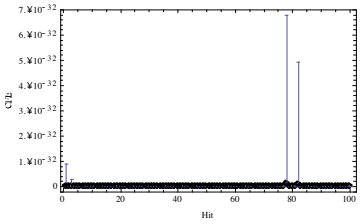
Algorithm 2



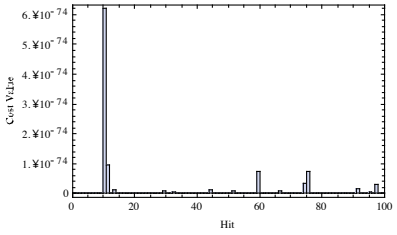


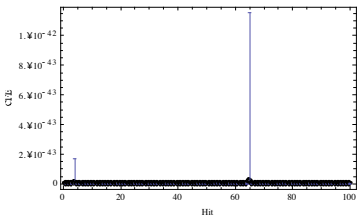
Algorithm 3



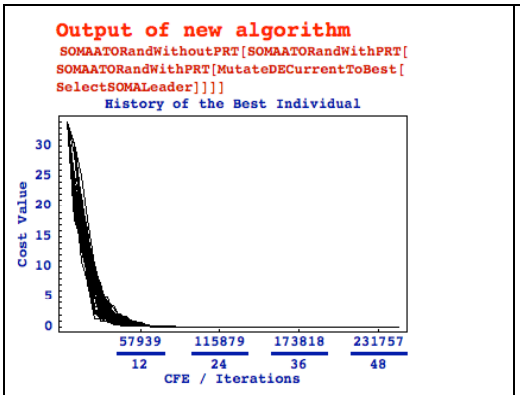


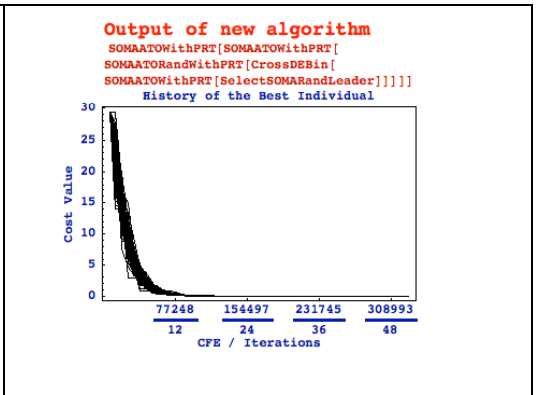
SOMAATO

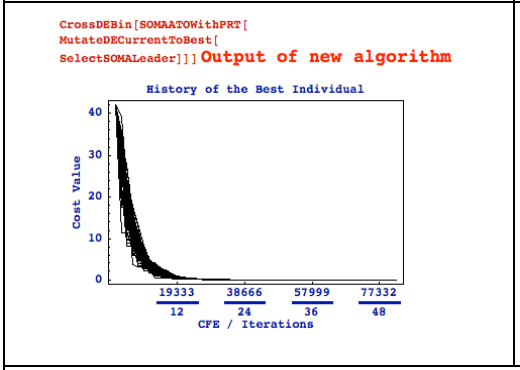


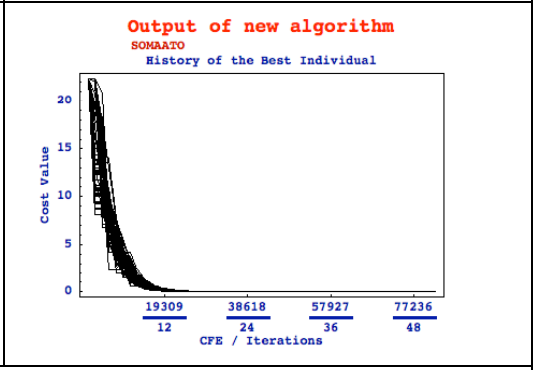


7.11. 4th De Jong's function – 20D

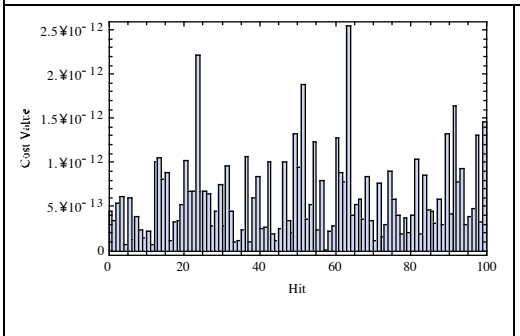


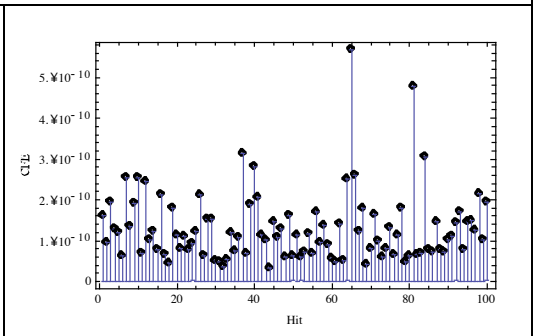




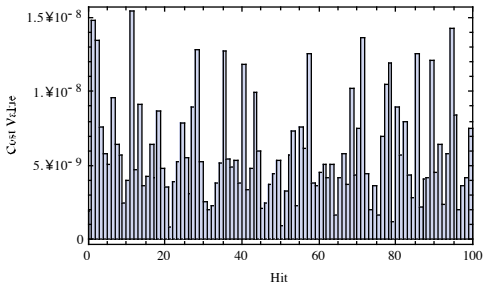


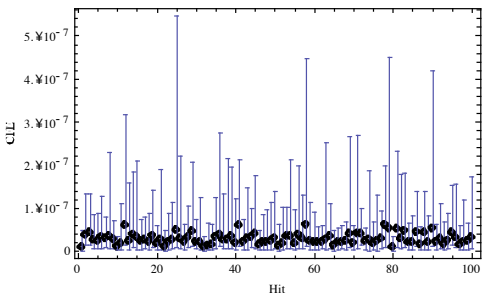
Algorithm 1



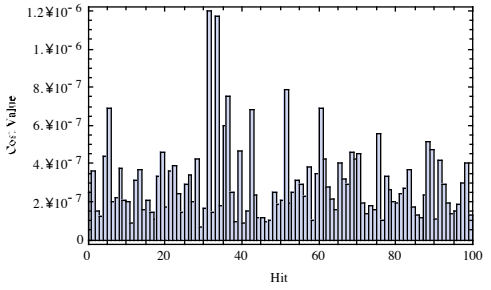


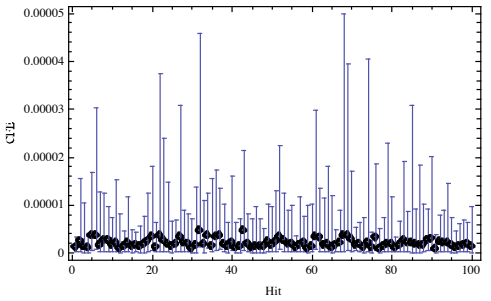
Algorithm 2



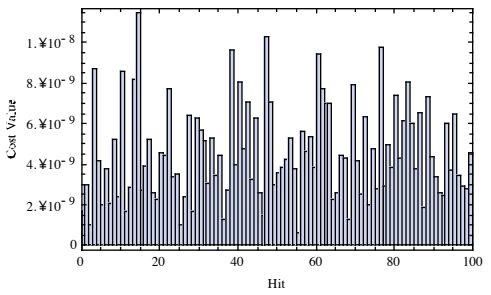


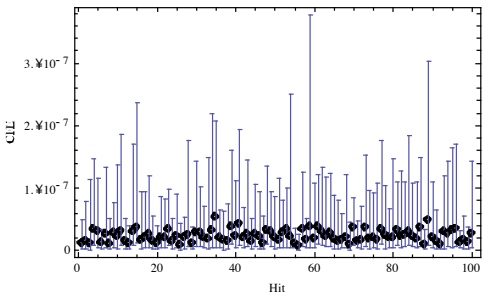
Algorithm 3



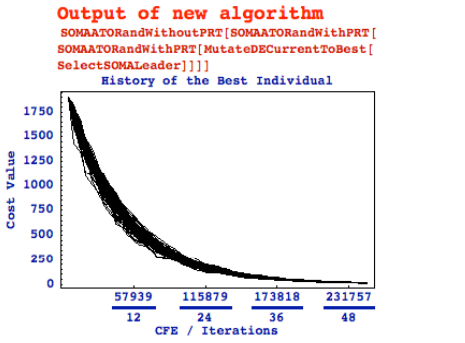


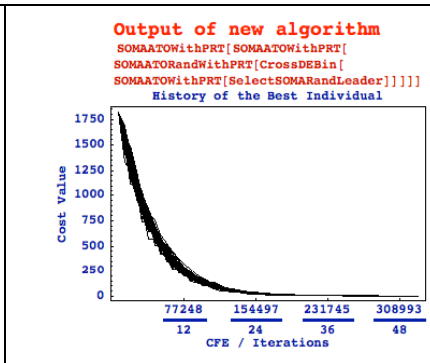
SOMAATO

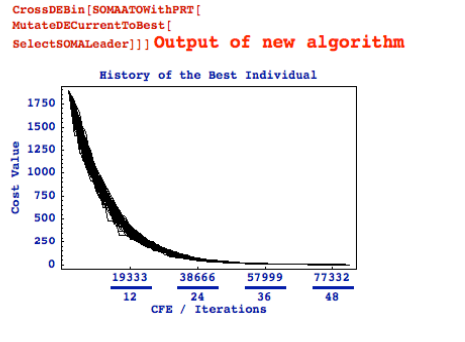


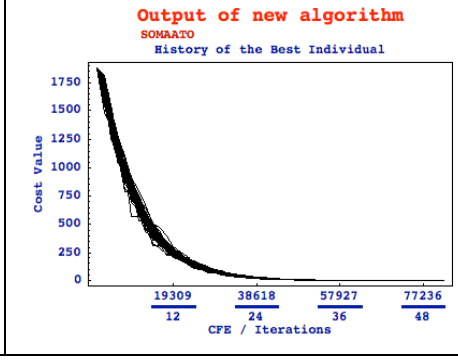


7.12. 4th De Jong's function – 100D

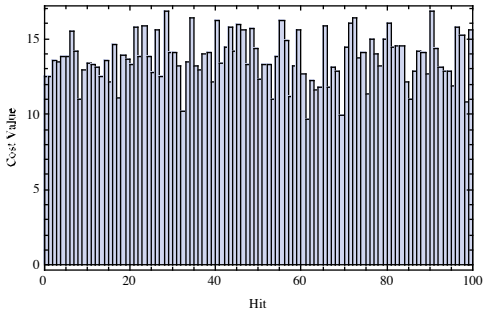


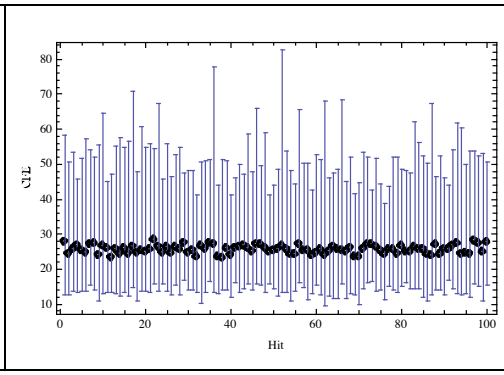




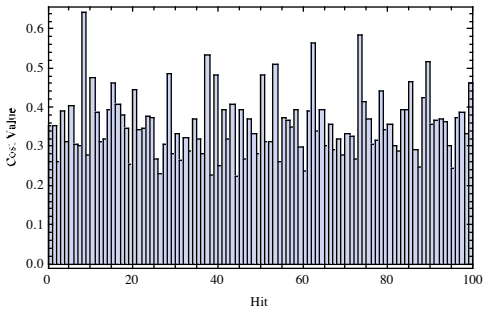


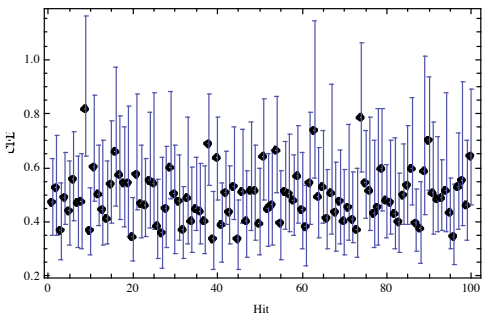
Algorithm 1



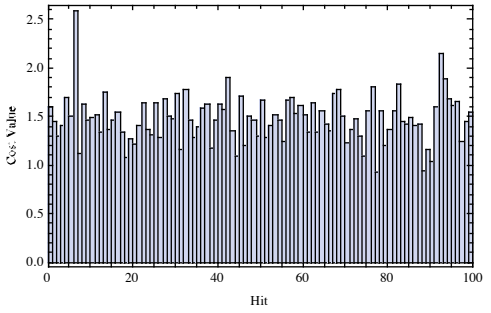


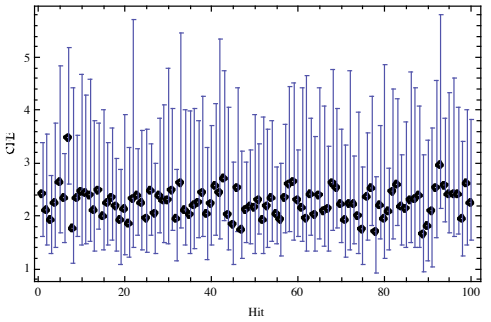
Algorithm 2



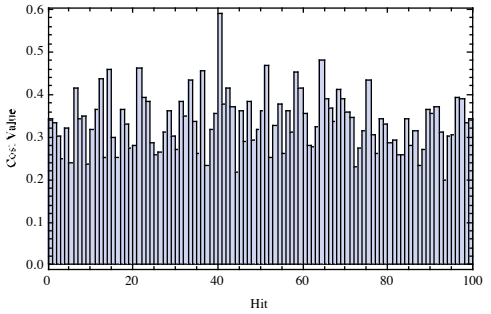


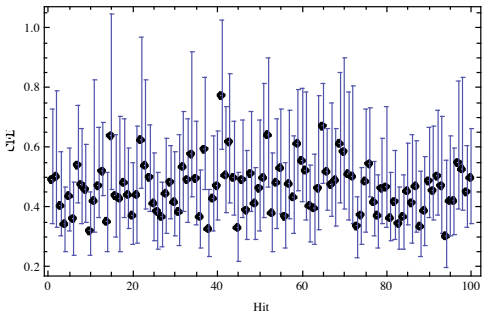
Algorithm 3





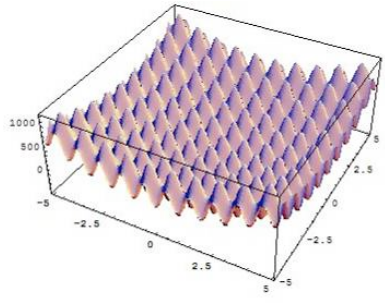
SOMAATO

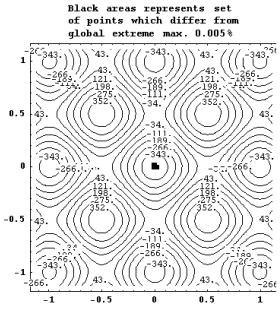


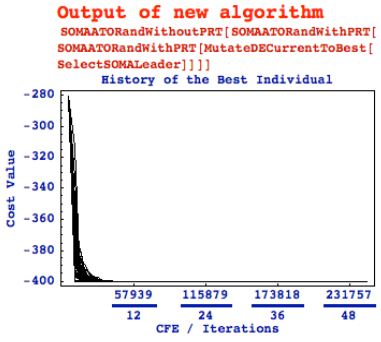


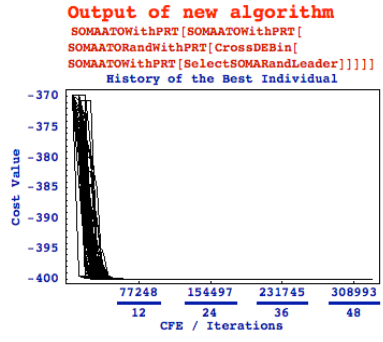
7.13. Rastrigin's function – 2D

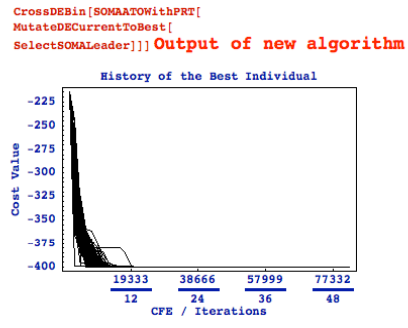
$$f(x) = 10Dim + \sum_{i=1}^{Dim}\left(x_i^{\,2} - 10\cos(2\pi x_i)\right)$$

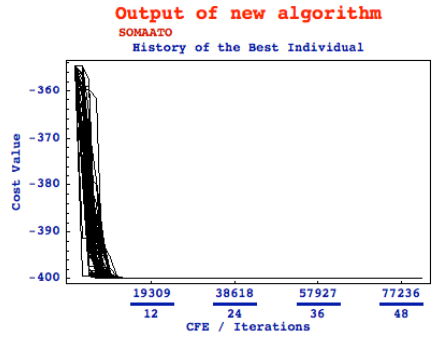


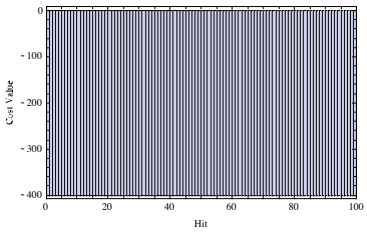
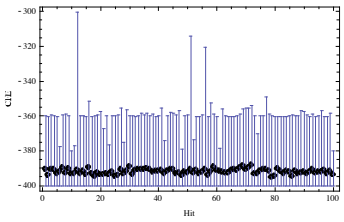
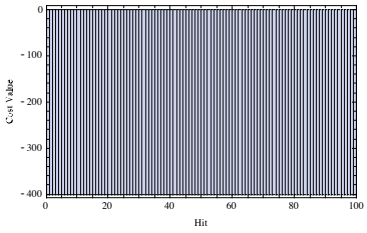
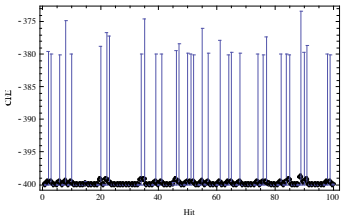
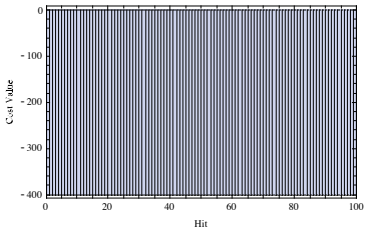
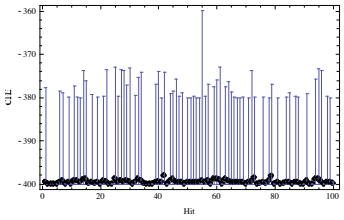
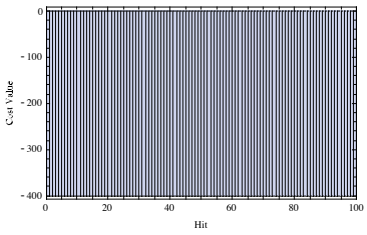
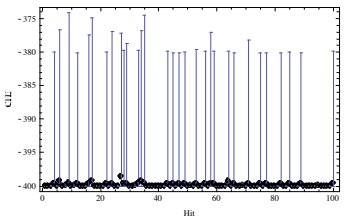




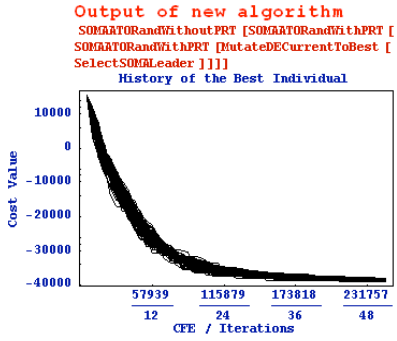


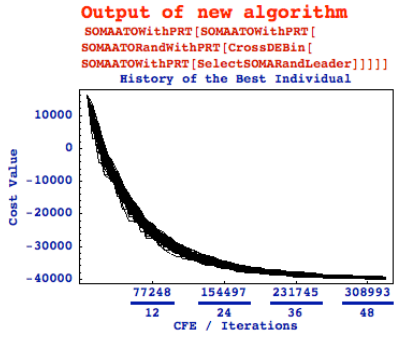


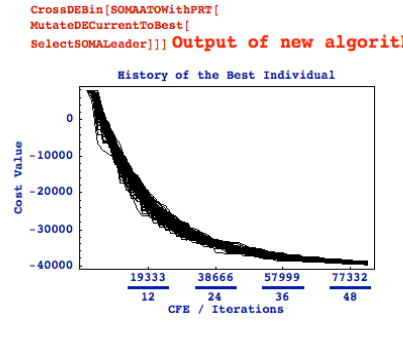


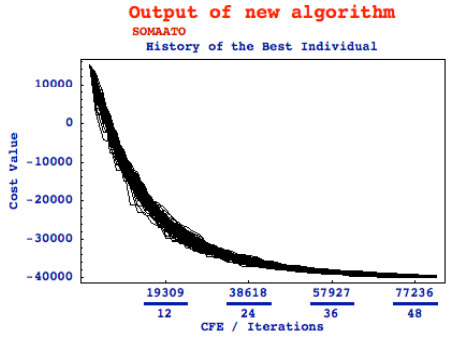
Algorithm 1	
	
Algorithm 2	
	
Algorithm 3	
	
SOMAATO	
	

7.14. Rastrigin's function – 20D

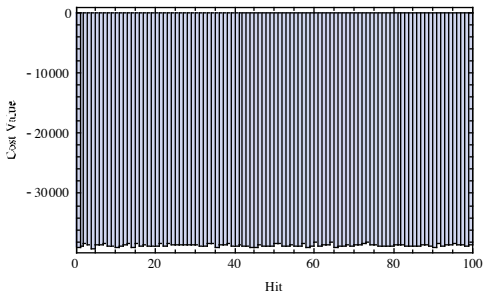


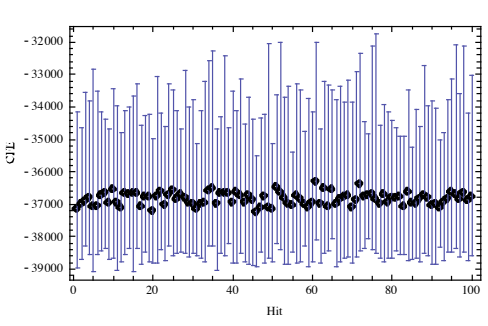




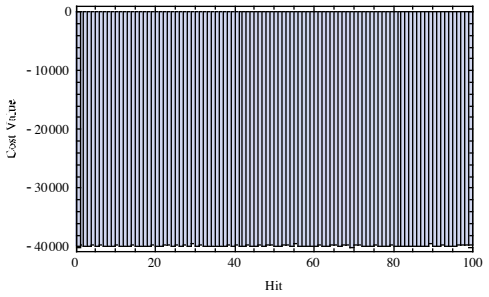


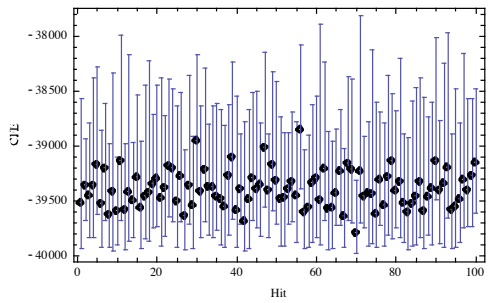
Algorithm 1



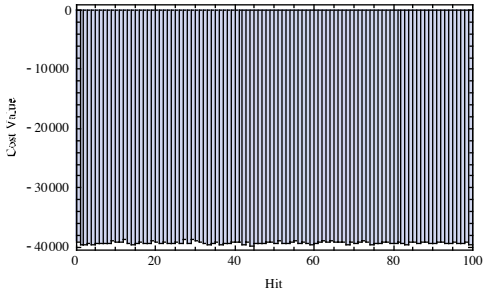


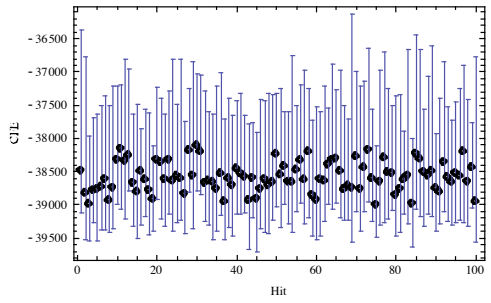
Algorithm 2



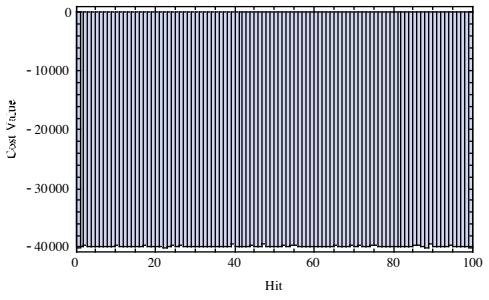


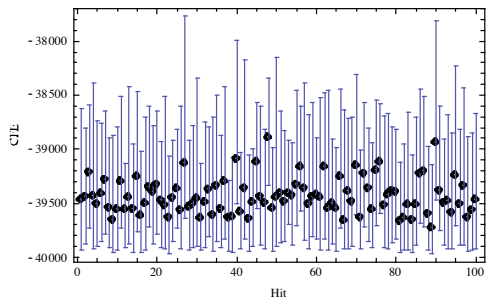
Algorithm 3



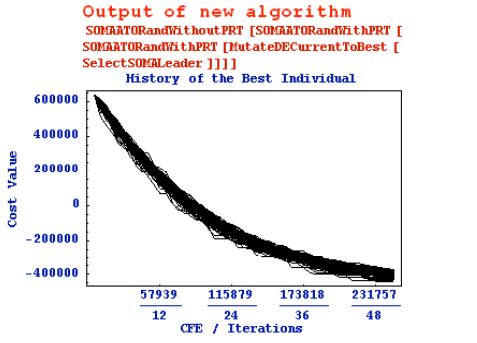


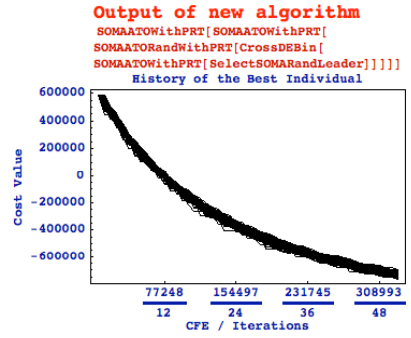
SOMAATO

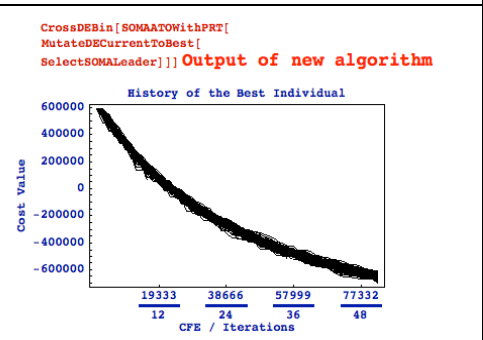


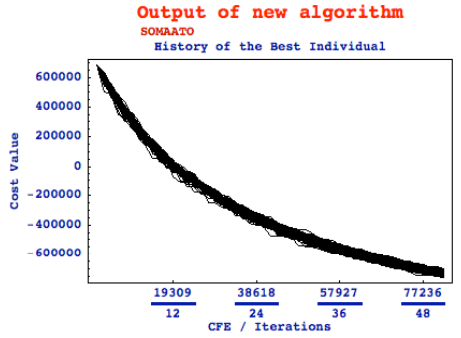


7.15. Rastrigin's function – 100D

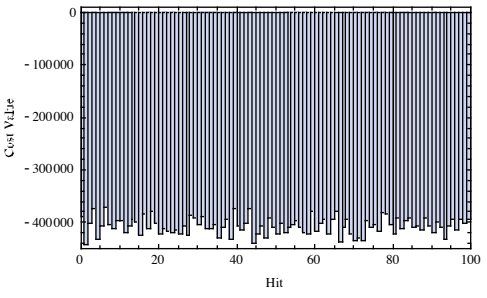


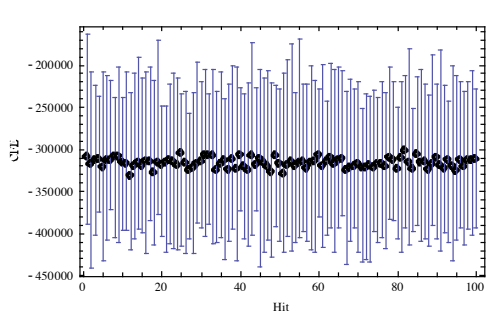




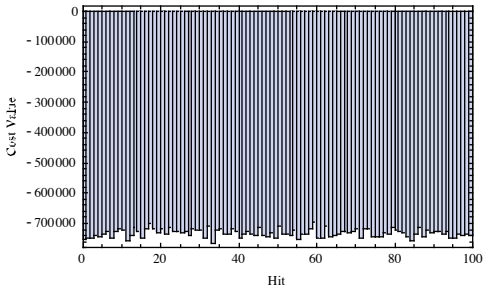


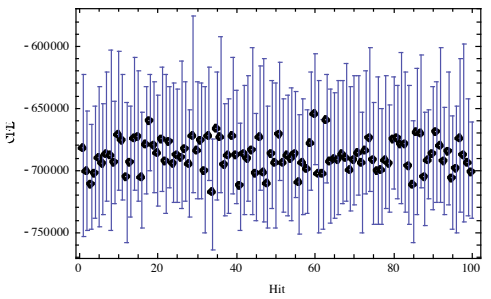
Algorithm 1



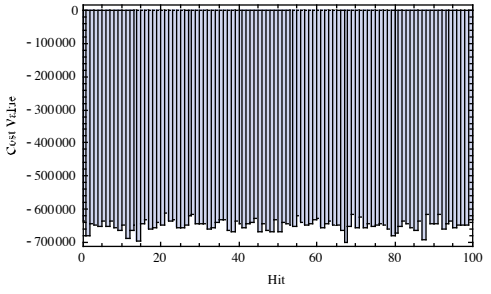


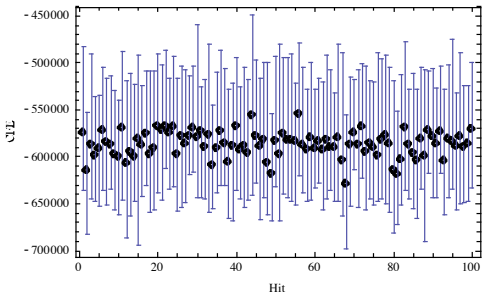
Algorithm 2



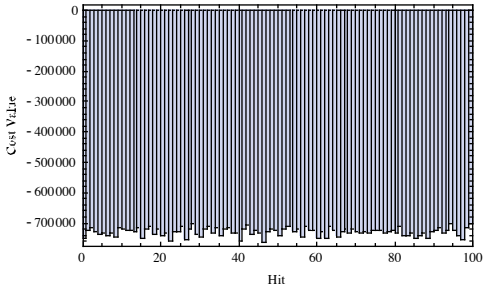


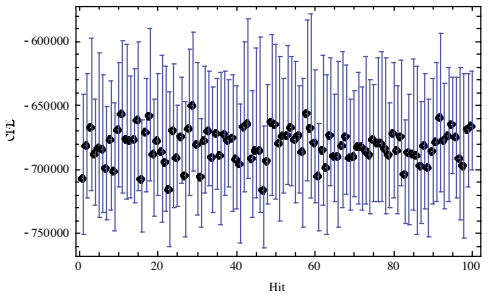
Algorithm 3





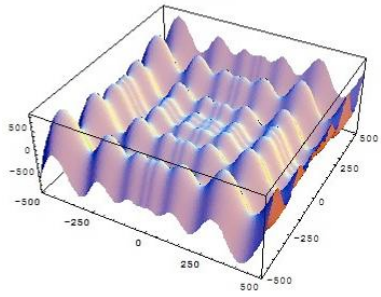
SOMAATO

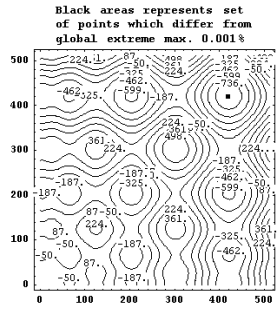


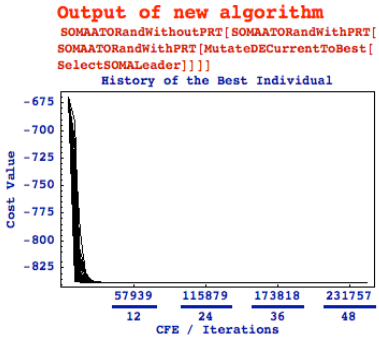


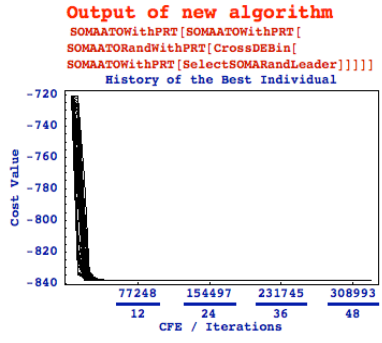
7.16. Schwefel's function – 2D

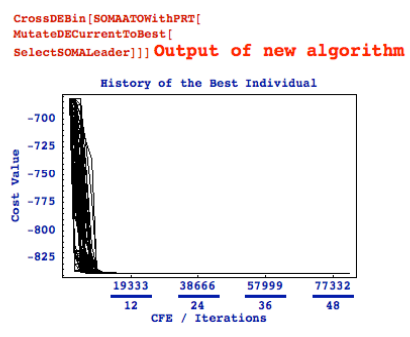
$$f(x) = \sum_{i=1}^{Dim} -x_i \cdot \sin\left(\sqrt{|x_i|}\right)$$

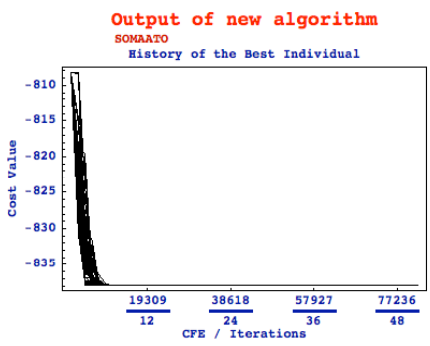


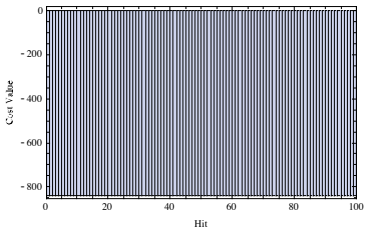
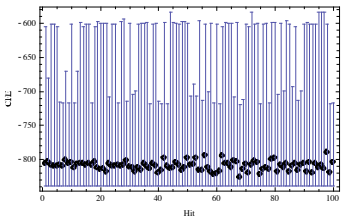
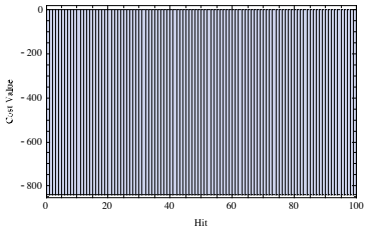
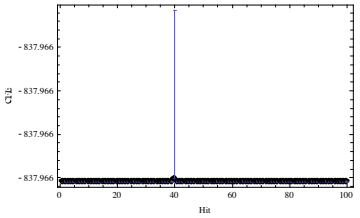
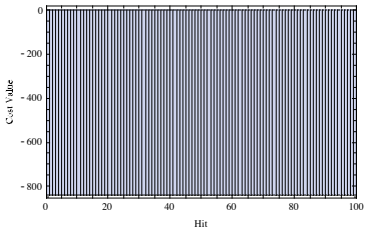
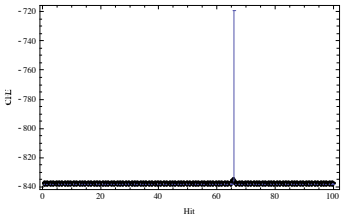
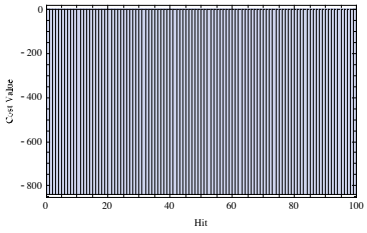
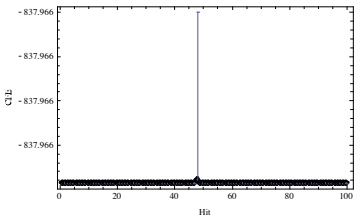




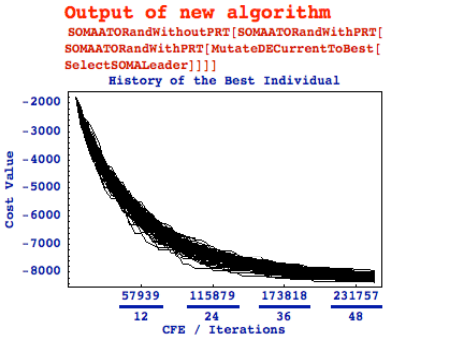


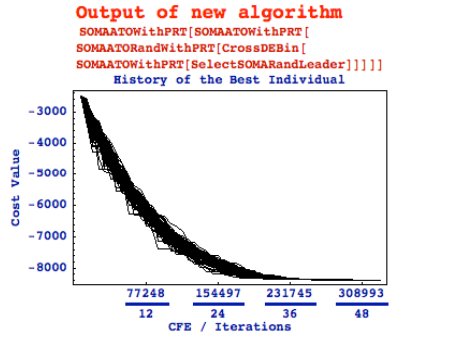


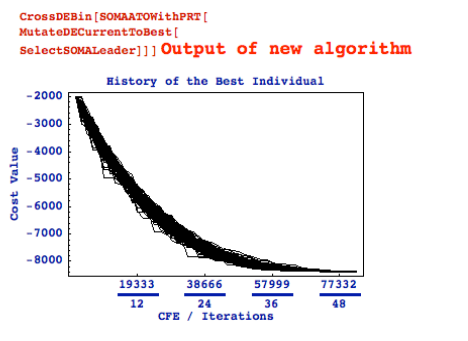


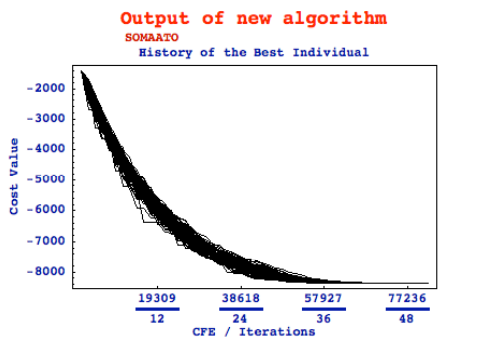
Algorithm 1	
	
Algorithm 2	
	
Algorithm 3	
	
SOMAATO	
	

7.17. Schwefel's function – 20D

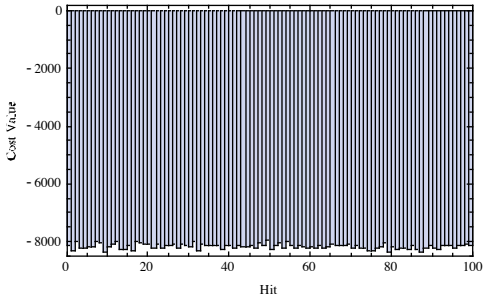


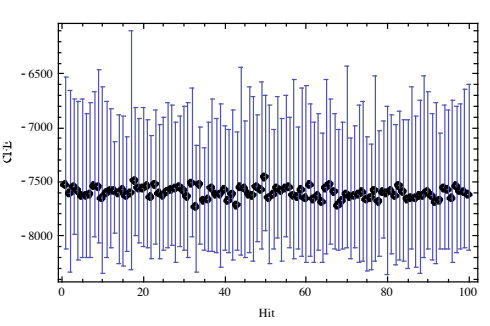




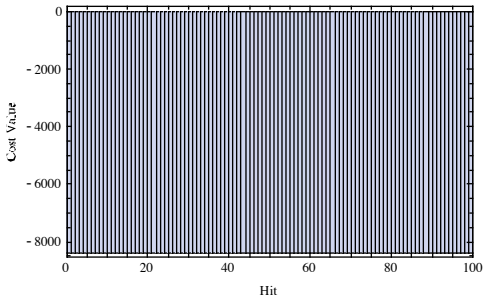


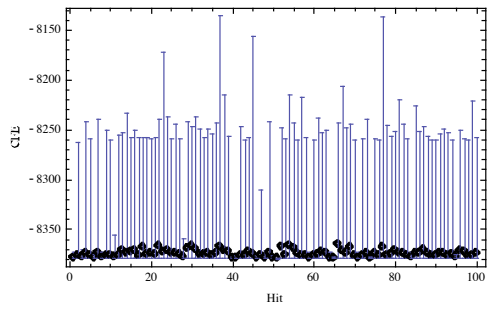
Algorithm 1



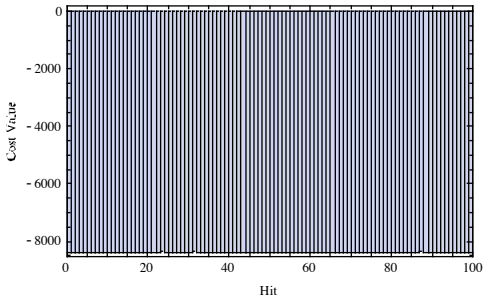


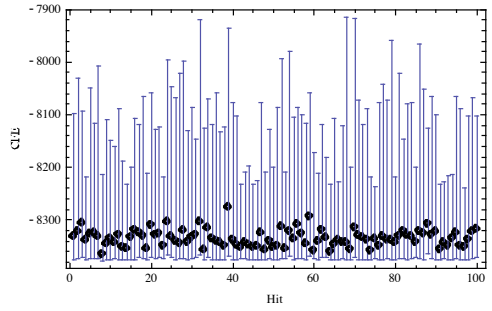
Algorithm 2



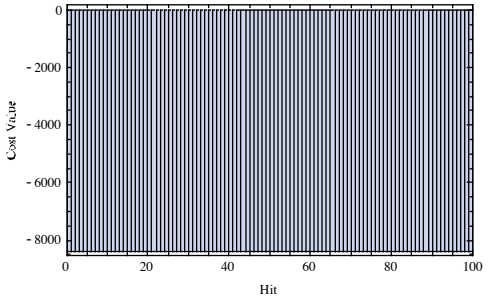


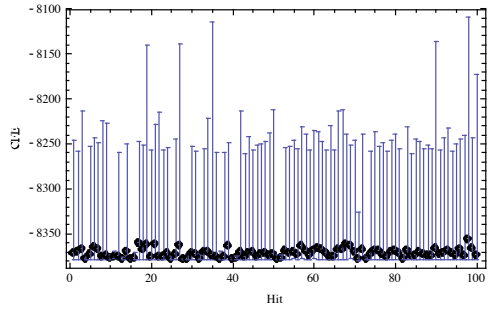
Algorithm 3



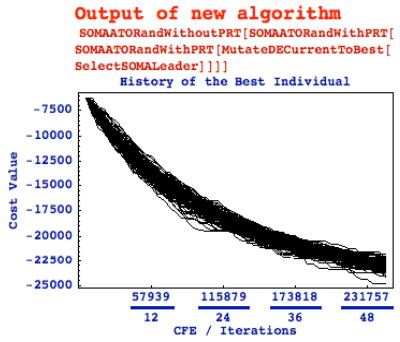


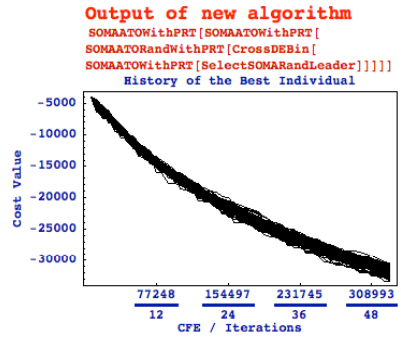
SOMAATO

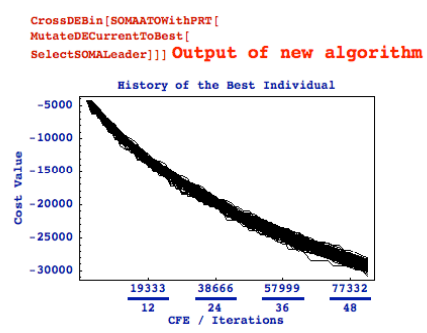


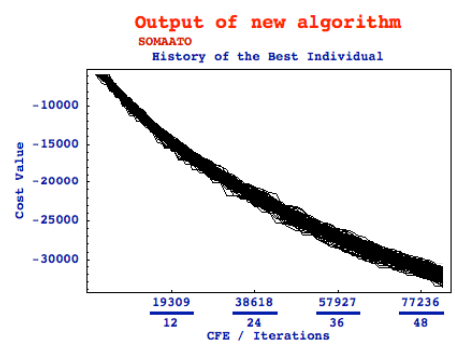


7.18. Schwefel's function – 100D

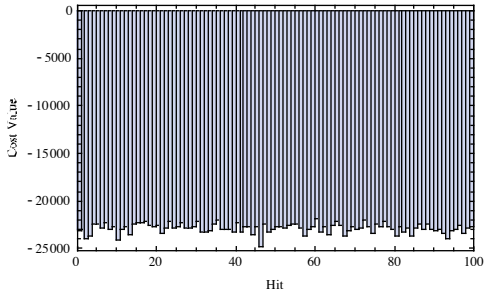


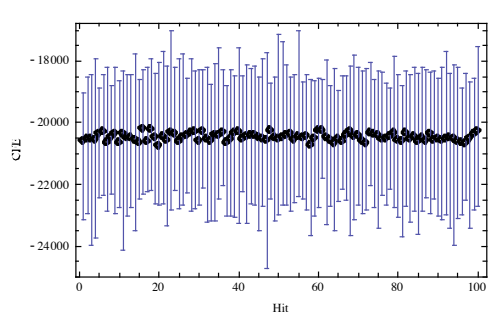




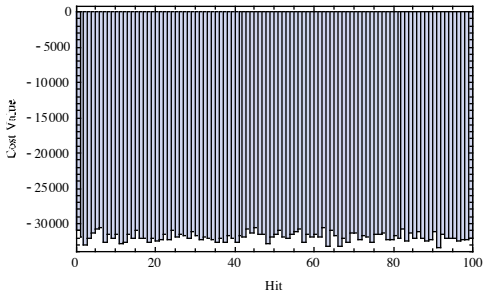


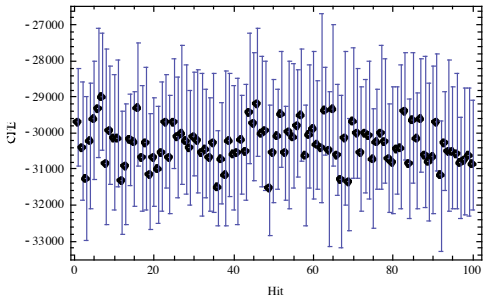
Algorithm 1



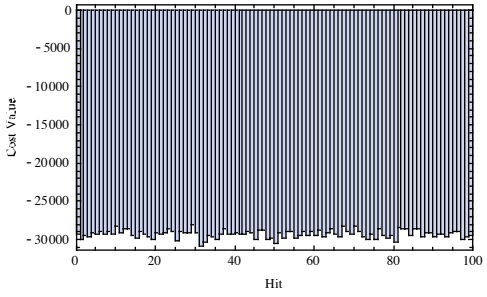


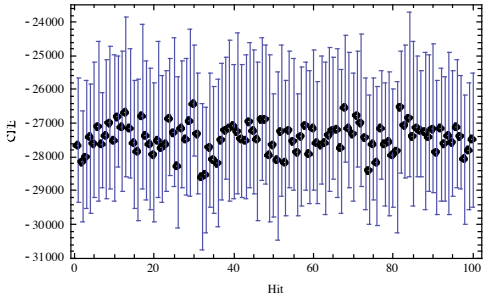
Algorithm 2



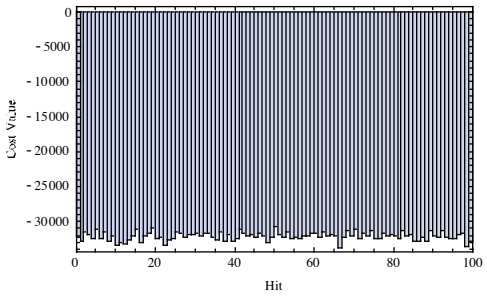


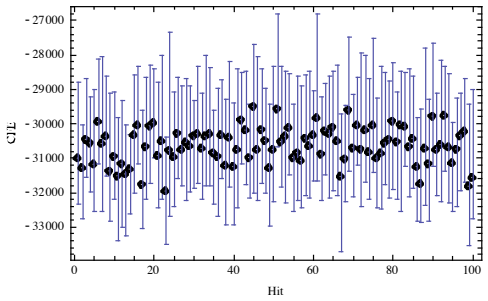
Algorithm 3





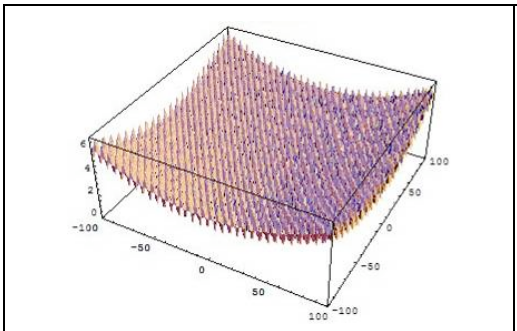
SOMAATO

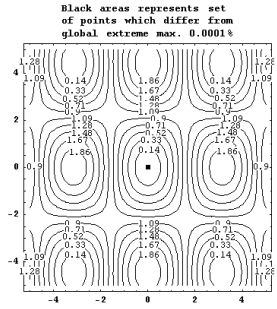


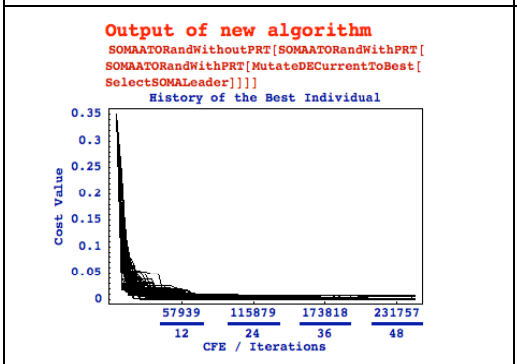


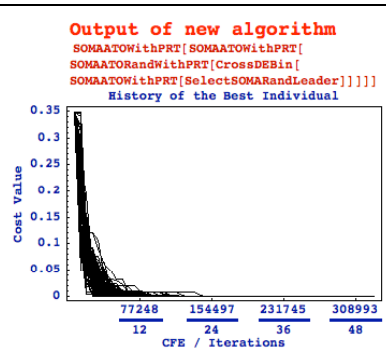
7.19. Griewangk's function – 2D

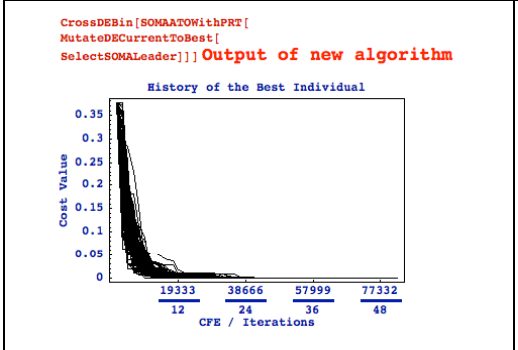
$$f(x) = \sum_{i=1}^{Dim} \frac{x_i^{\,2}}{4000} - \prod_{i=1}^{Dim} \cos\left(\frac{x_i}{\sqrt{i}}\right) + 1$$

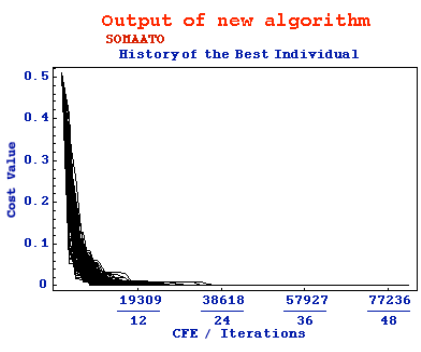


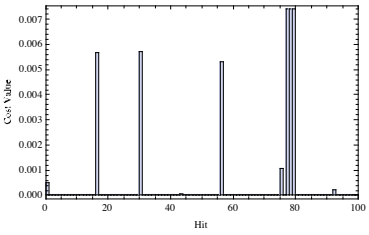
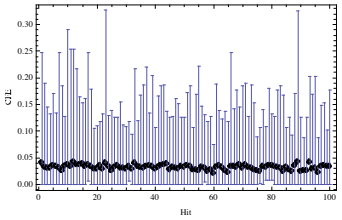
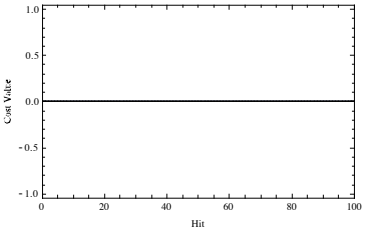
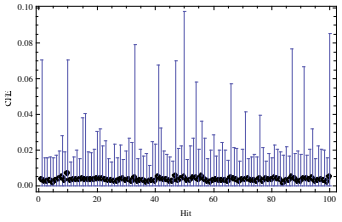
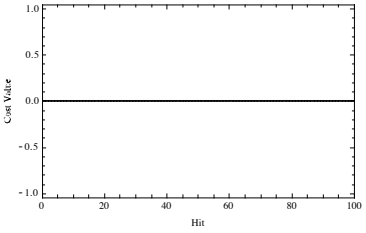
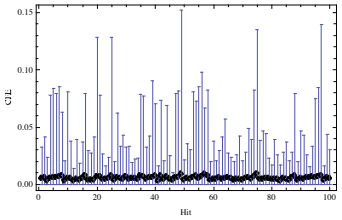
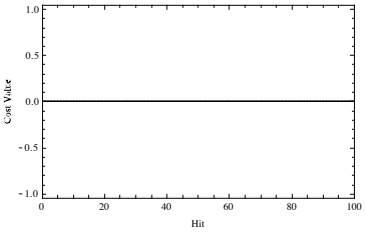
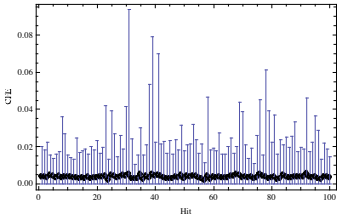




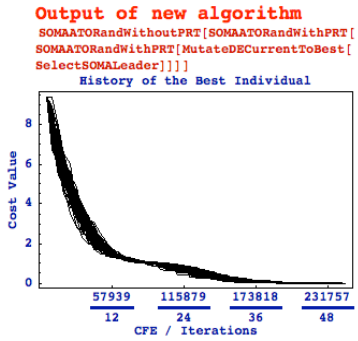


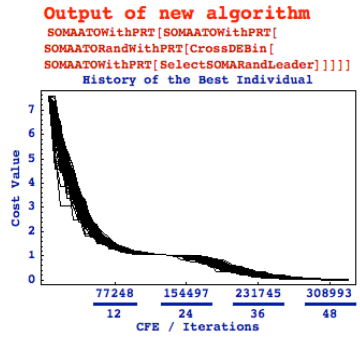


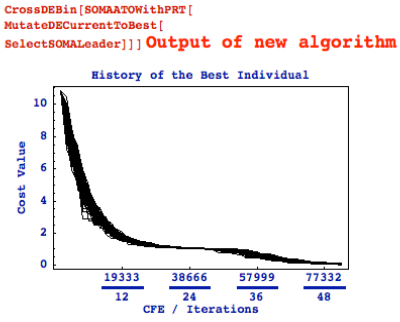


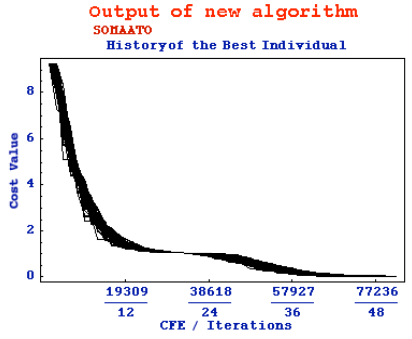
Algorithm 1	
	
Algorithm 2	
	
Algorithm 3	
	
SOMAATO	
	

7.20. Griewangk's function – 20D

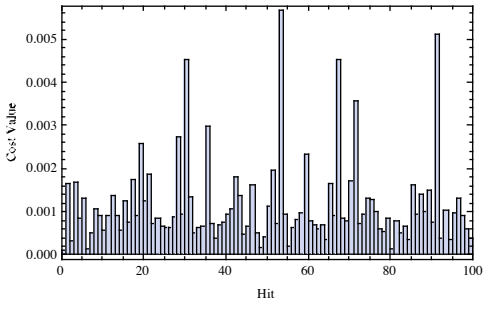


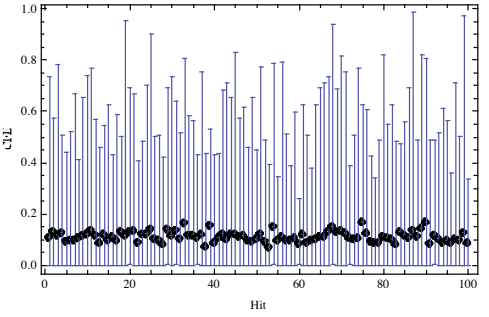




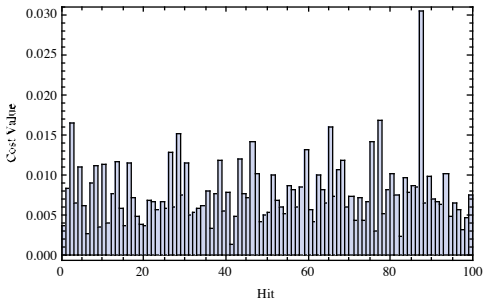


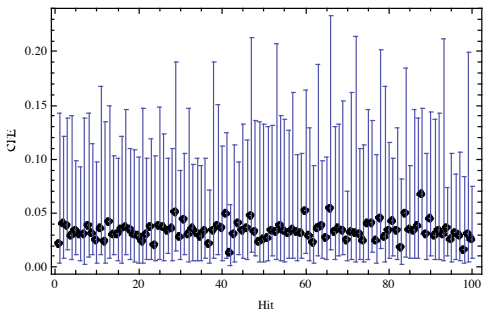
Algorithm 1



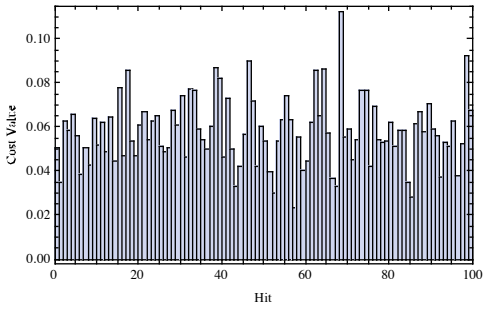


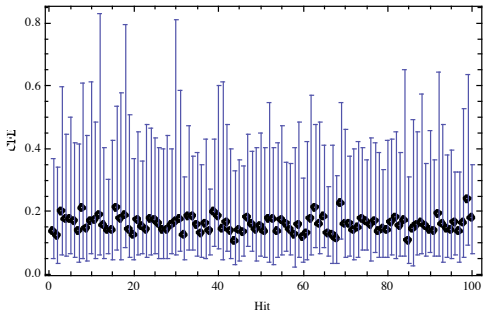
Algorithm 2



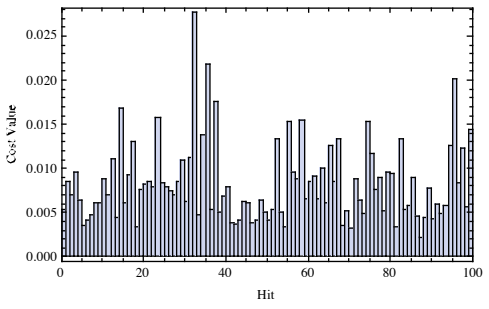


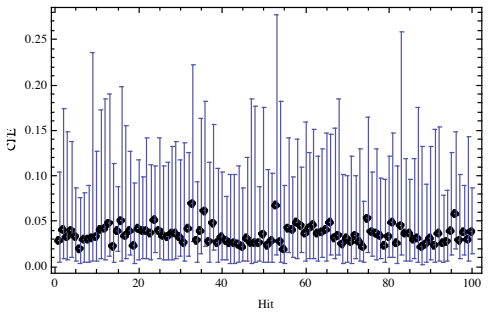
Algorithm 3



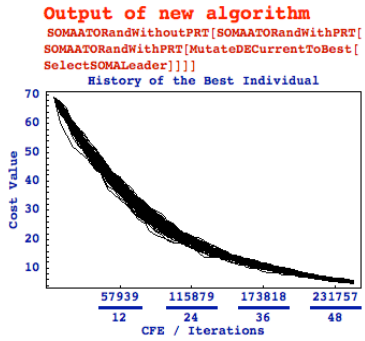


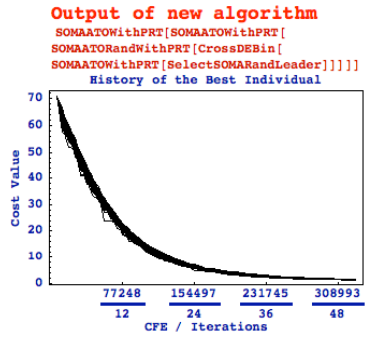
SOMAATO

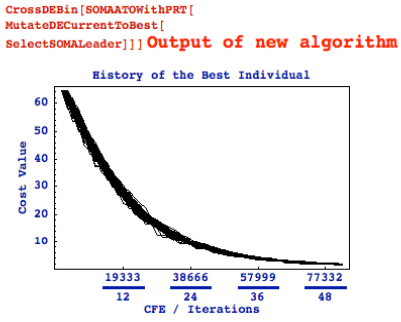


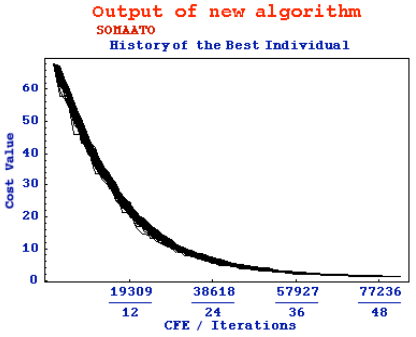


7.21. Griewangk's function – 100D

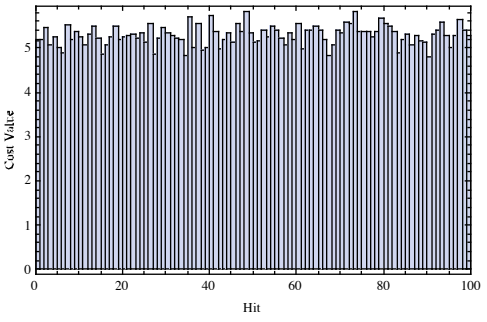


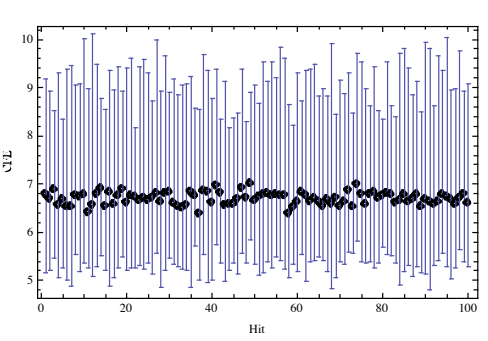




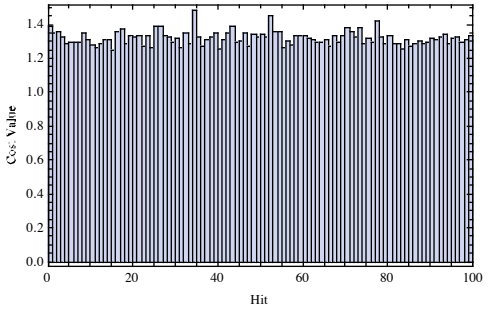


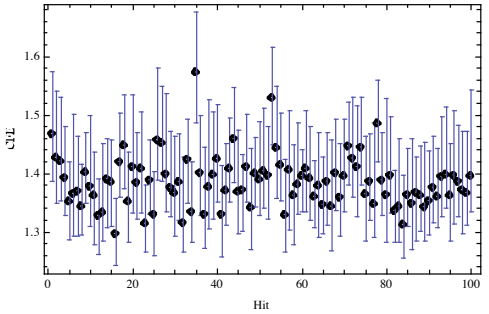
Algorithm 1



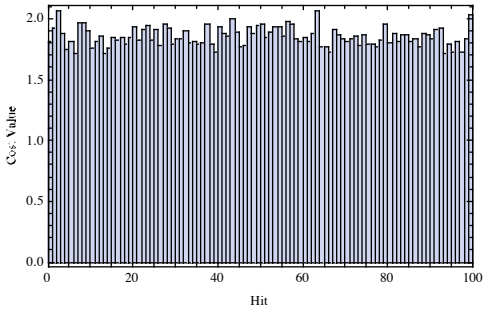


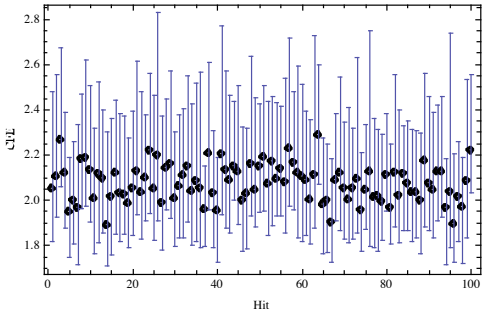
Algorithm 2



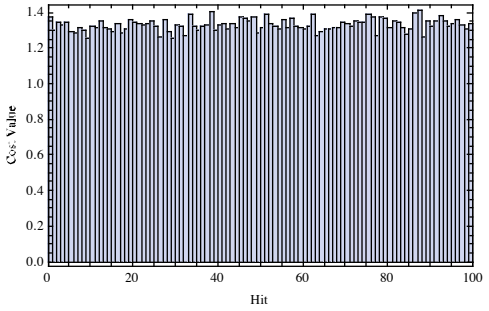


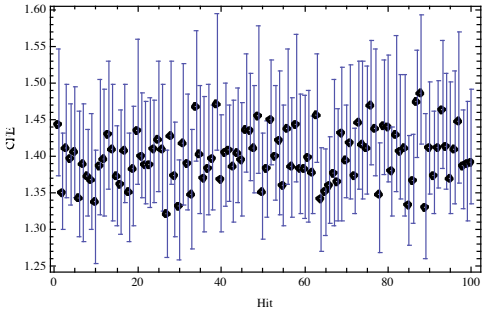
Algorithm 3





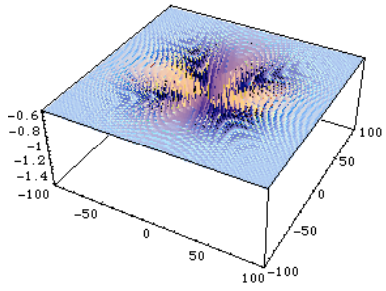
SOMAATO

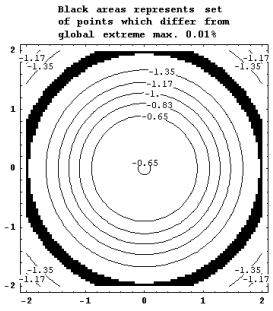


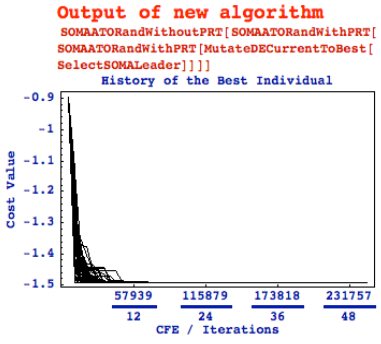


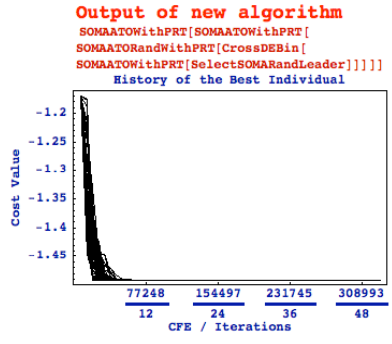
7.22. Sine envelope sine wave function – 2D

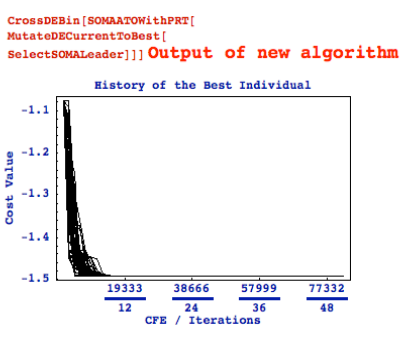
$$f(x) = -\sum_{i=1}^{Dim-1}\left(\frac{\sin^2\left(\sqrt{x_{i+1}^2 + x_i^2}\right) - 0.5}{\left(0.001\left(x_{i+1}^2 + x_i^2\right) + 1\right)^2} + 0.5\right)$$

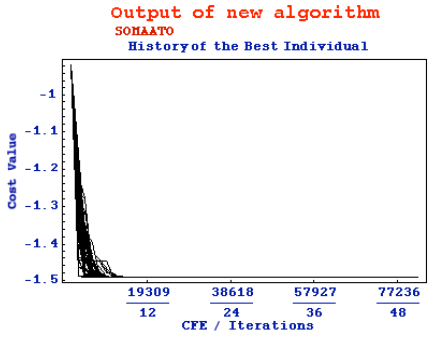


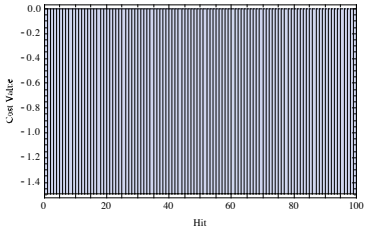
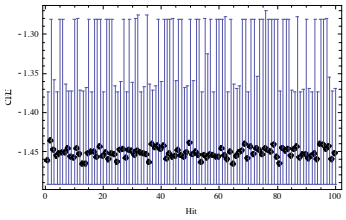
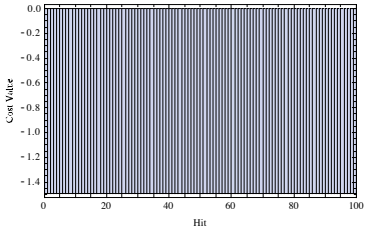
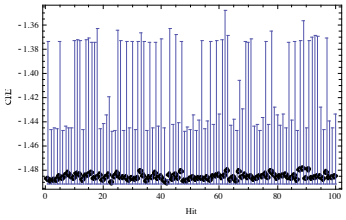
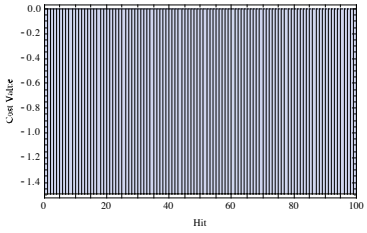
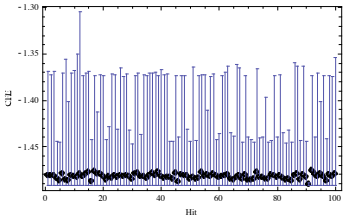
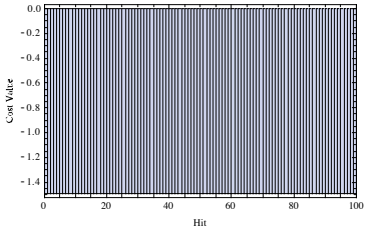
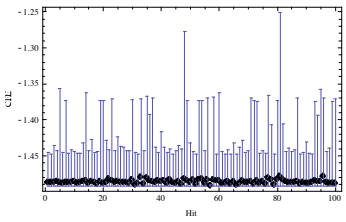




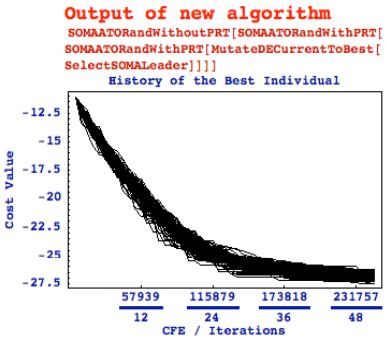


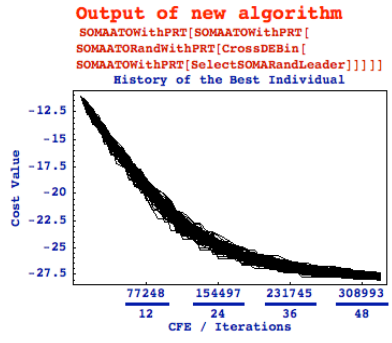


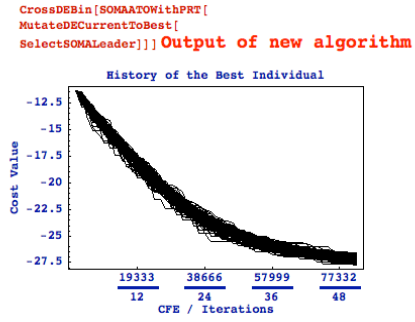


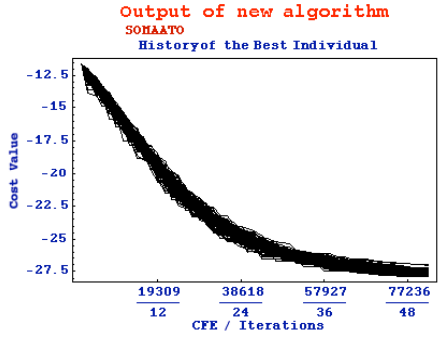
Algorithm 1	
	
Algorithm 2	
	
Algorithm 3	
	
SOMAATO	
	

7.23. Sine envelope sine wave function – 20D

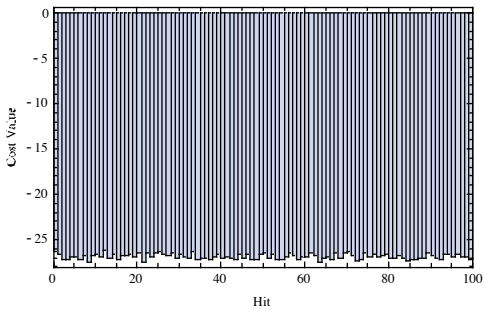


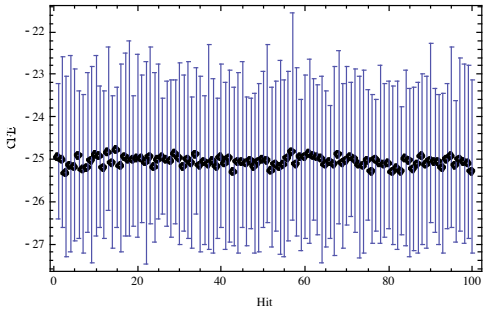




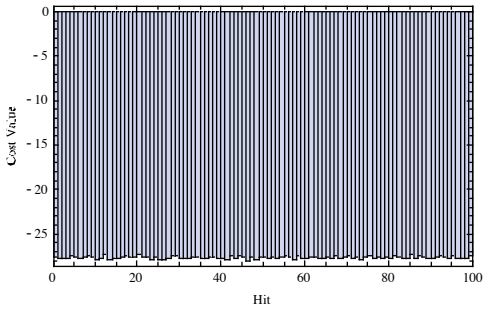


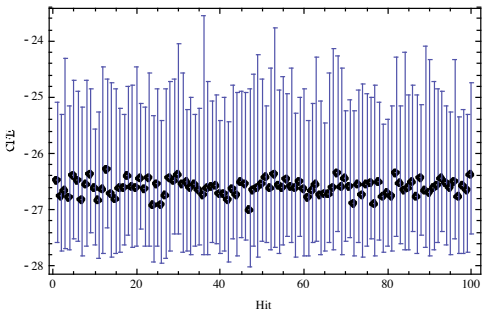
Algorithm 1



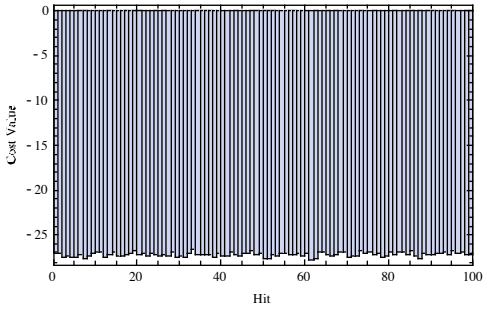


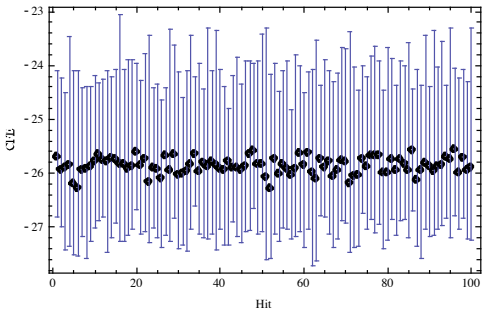
Algorithm 2



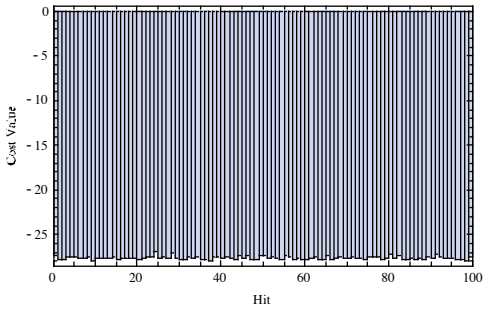


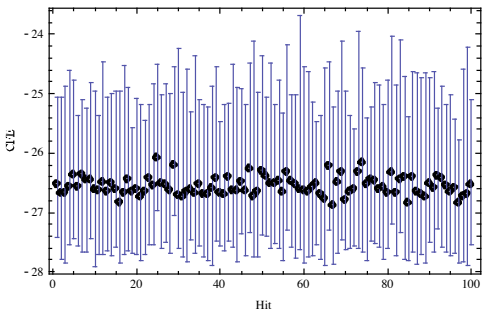
Algorithm 3



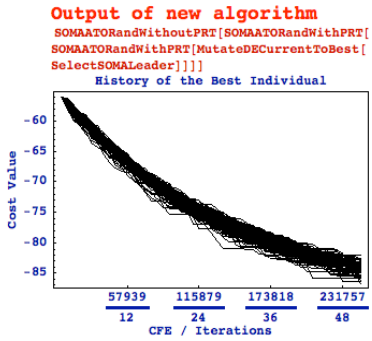


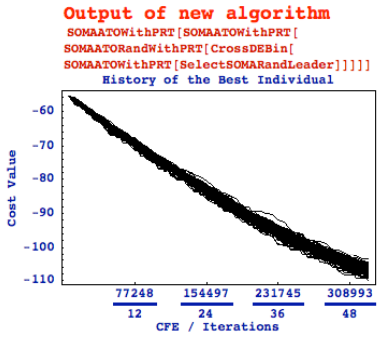
SOMAATO

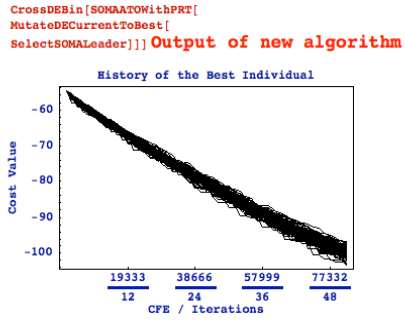


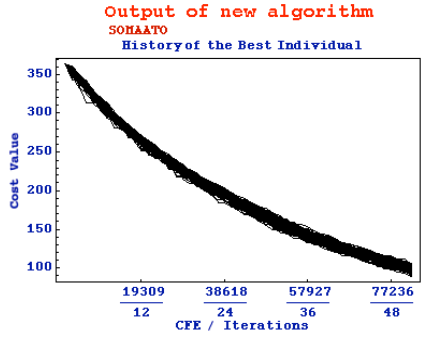


7.24. Sine envelope sine wave function – 100D

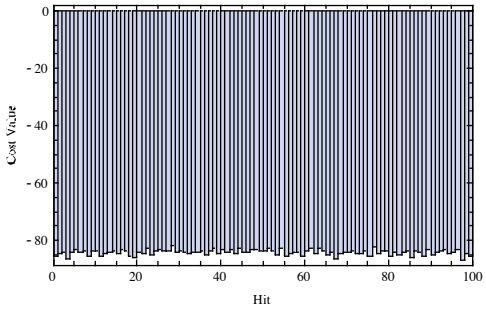


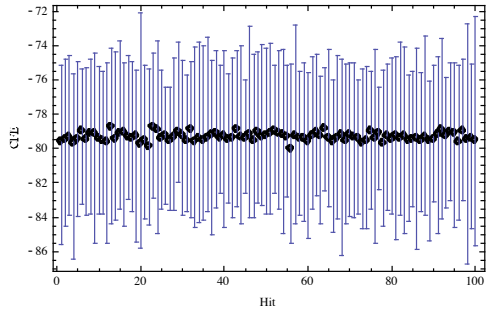




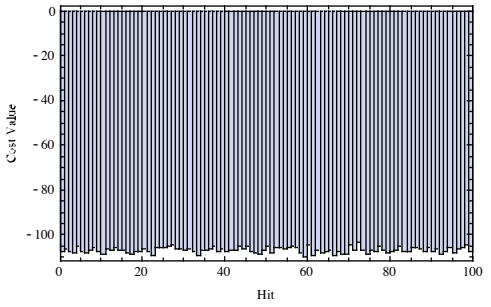


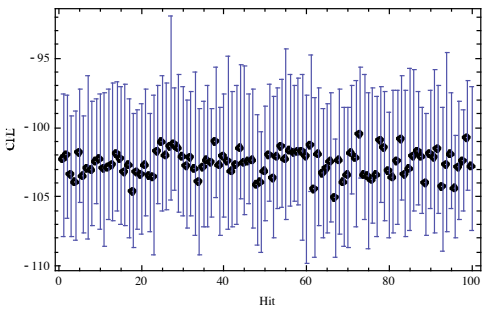
Algorithm 1



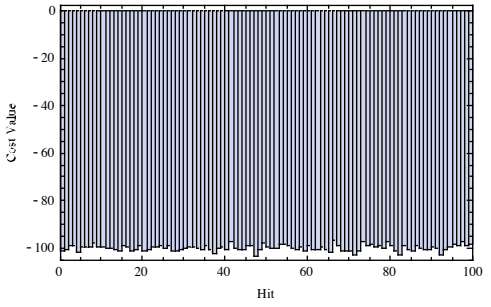


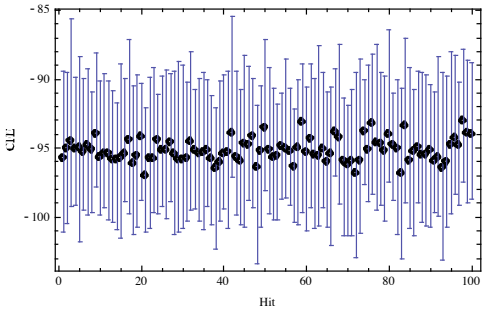
Algorithm 2



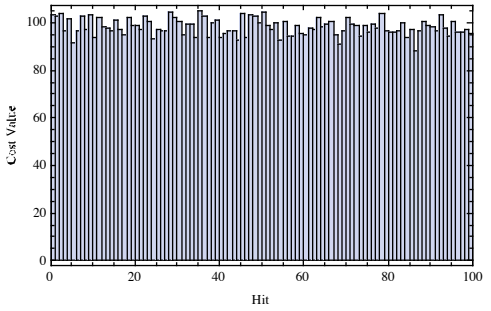


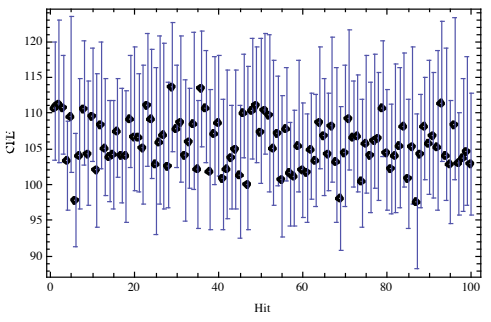
Algorithm 3





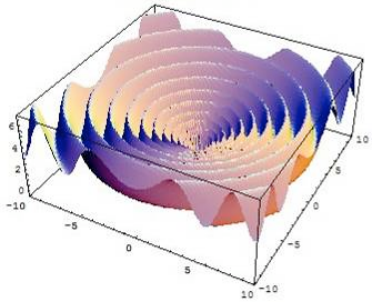
SOMAATO

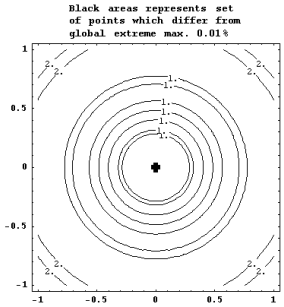


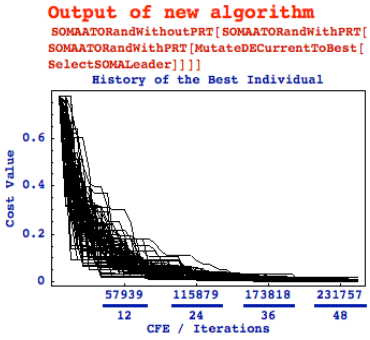


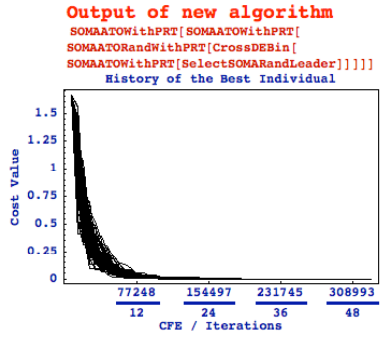
7.25. Stretched V sine wave function (Ackley) – 2D

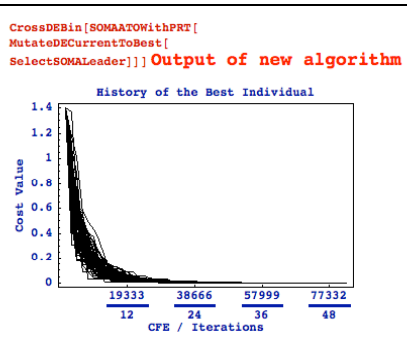
$$f(x) = \sum_{i=1}^{Dim-1} \left(x_{i+1}^{\,2} + x_i^{\,2} \right)^{0.25} \left(\sin^2\left(50 \left(x_{i+1}^{\,2} + x_i^{\,2} \right)^{0.1} \right) + 1 \right)$$

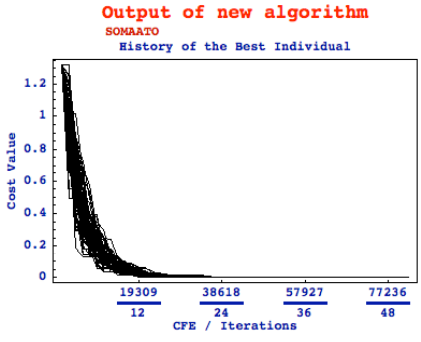




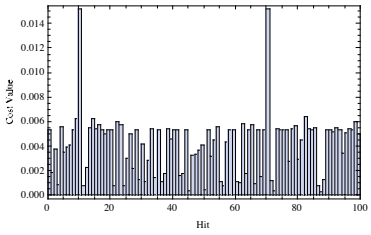


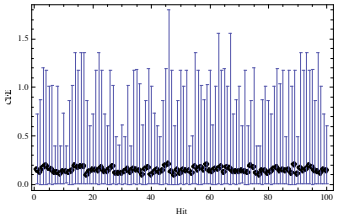




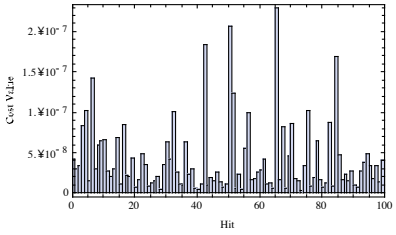


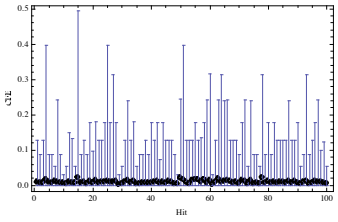
Algorithm 1



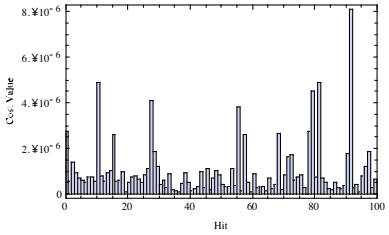


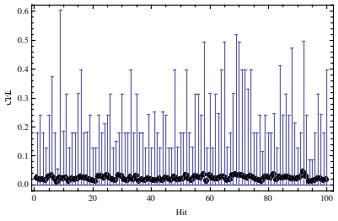
Algorithm 2



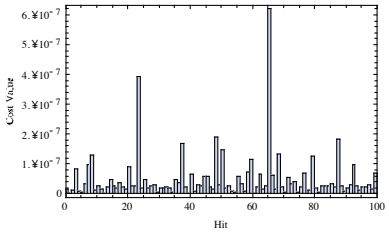


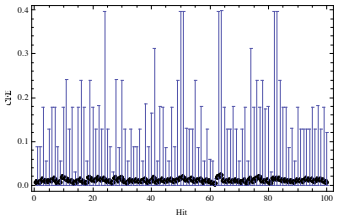
Algorithm 3



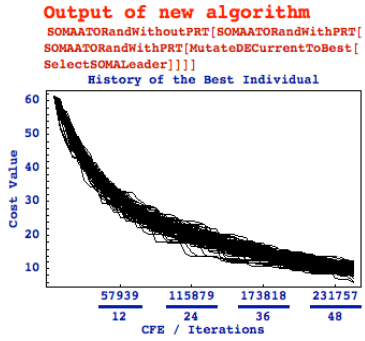


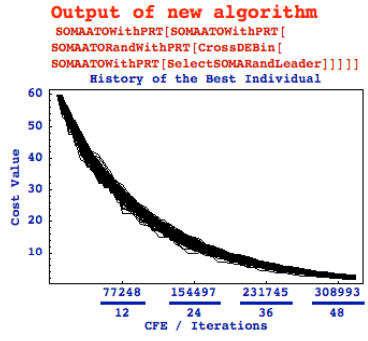
SOMAATO

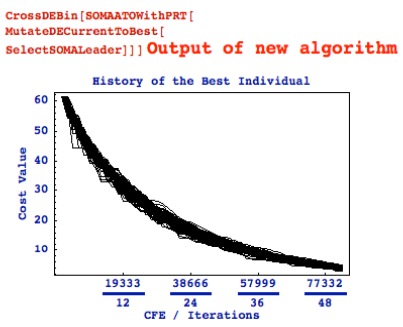


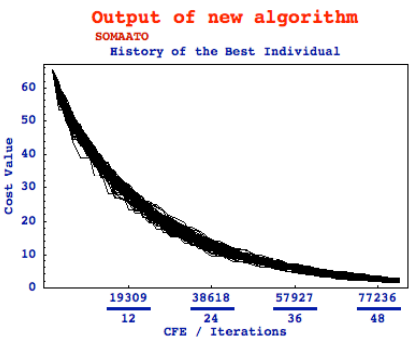


7.26. Stretched V sine wave function (Ackley) – 20D

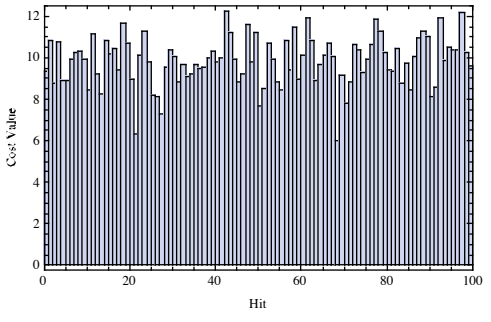


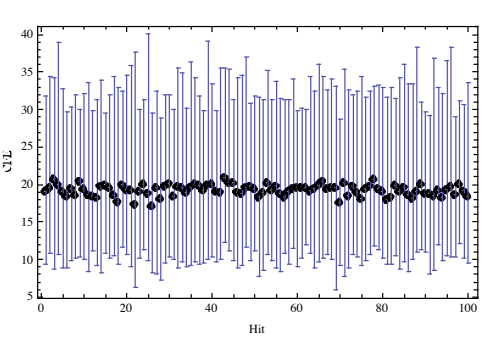




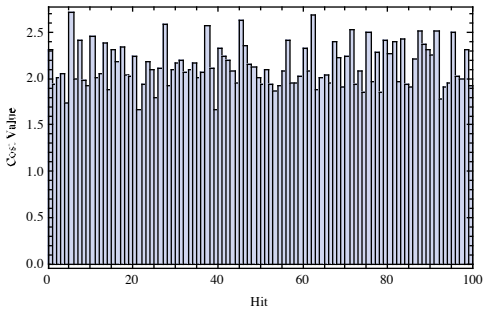


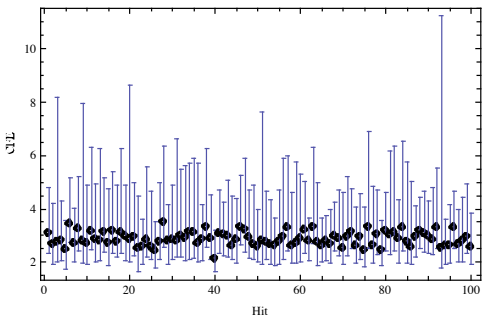
Algorithm 1



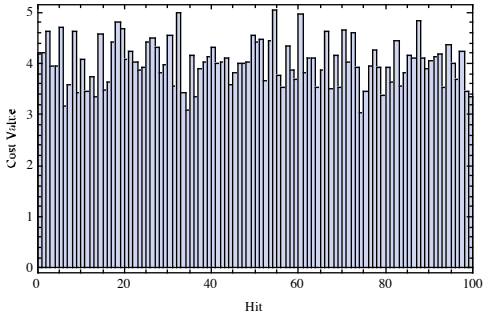


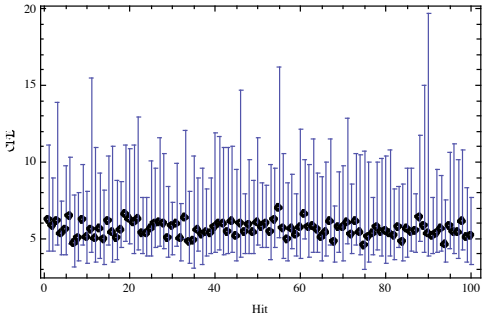
Algorithm 2



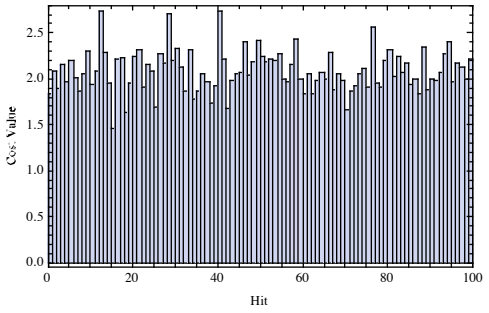


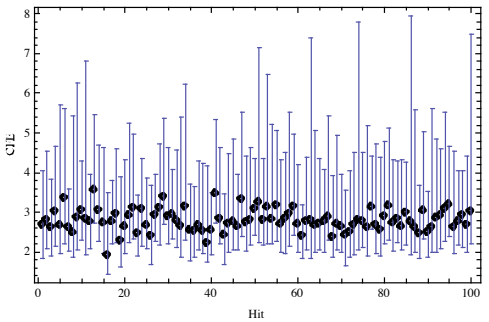
Algorithm 3



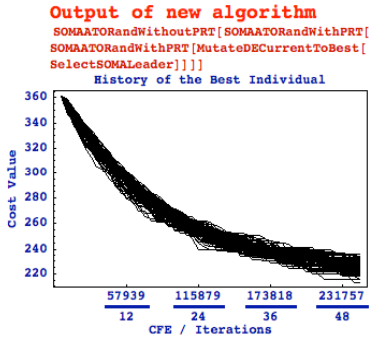


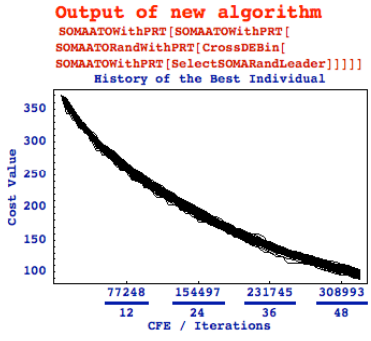
SOMAATO

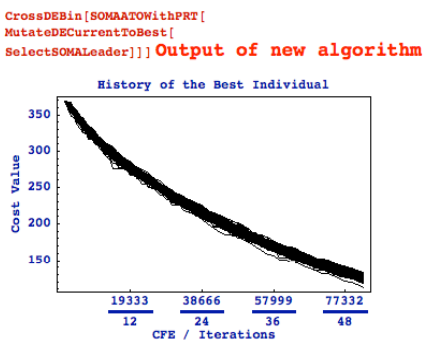


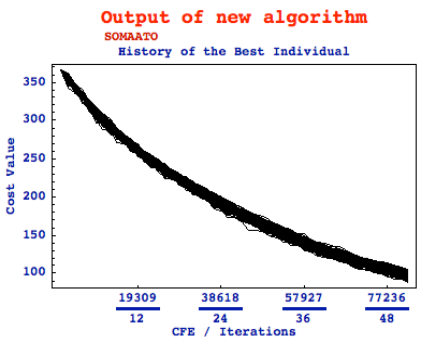


7.27. Stretched V sine wave function (Ackley) – 100D

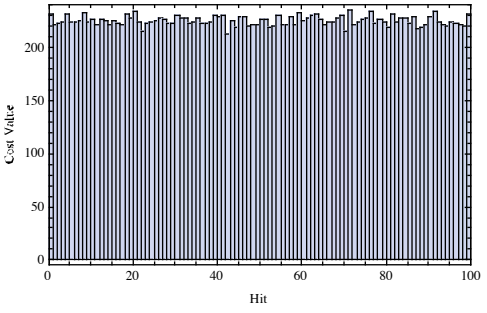


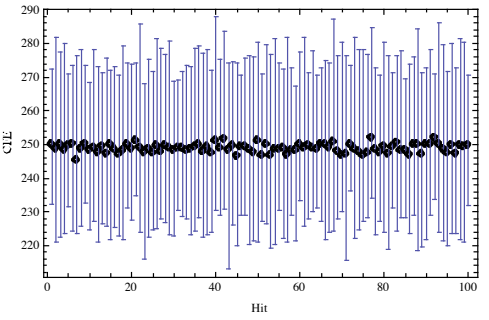




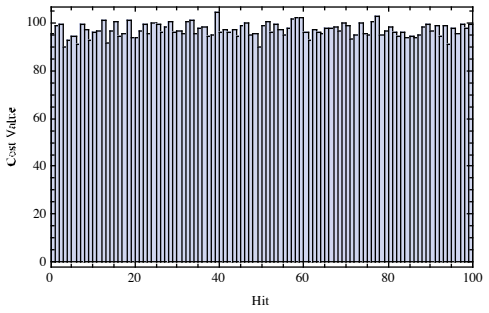


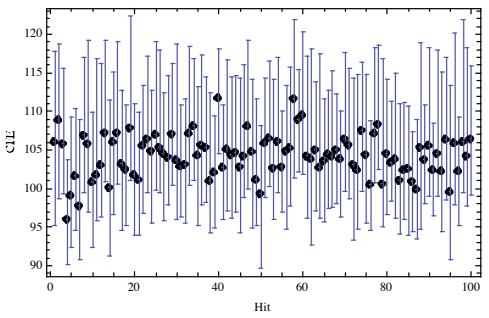
Algorithm 1



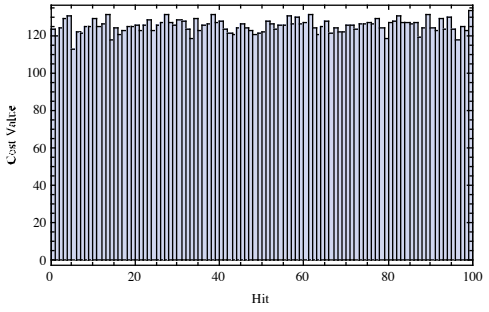


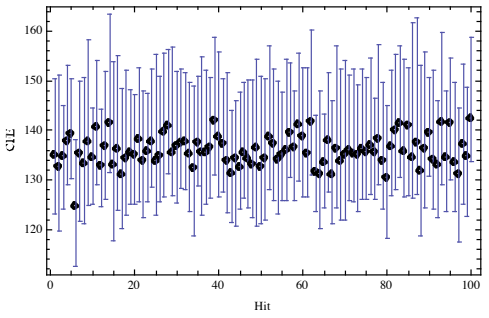
Algorithm 2



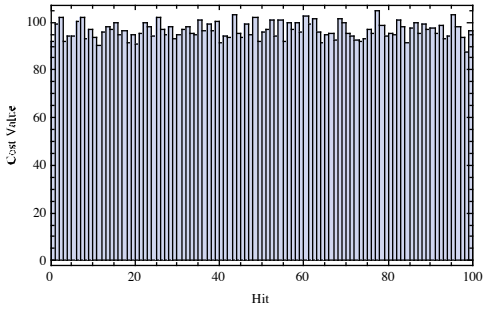


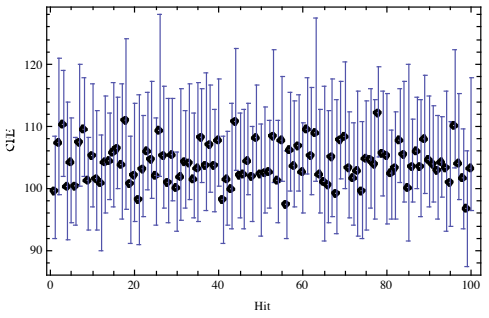
Algorithm 3





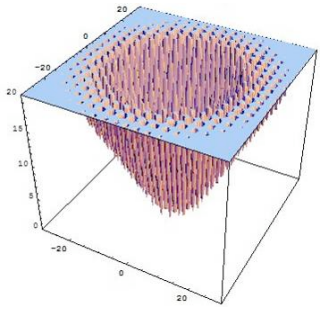
SOMAATO

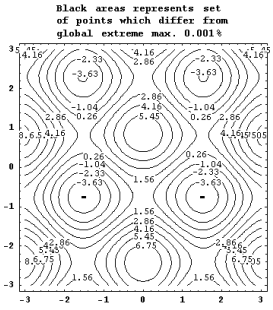


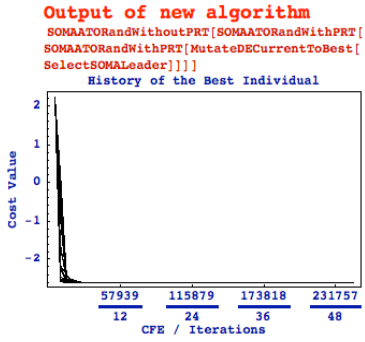


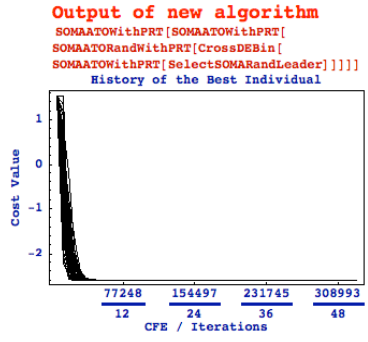
7.28. Test function (Ackley) – 2D

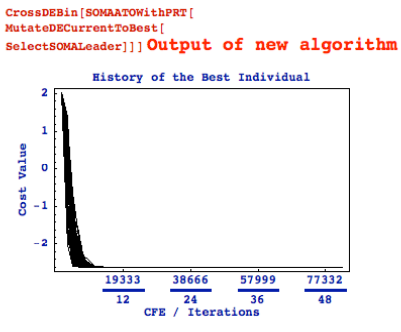
$$f(x) = \sum_{i=1}^{Dim-1} \left(3\left(\cos(2x_i) + \sin(2x_{i+1})\right) + \frac{\sqrt{x_{i+1}^{\,2} + x_i^{\,2}}}{e^{0.2}} \right)$$

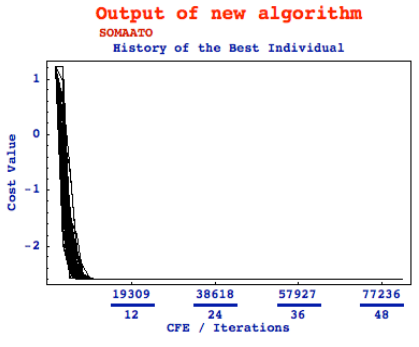


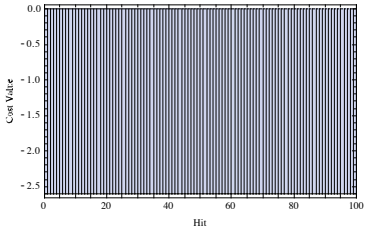
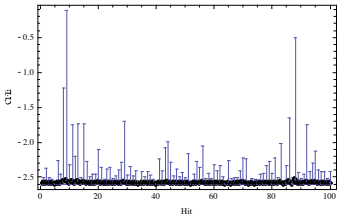
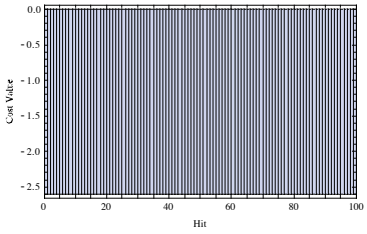
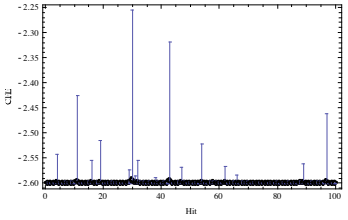
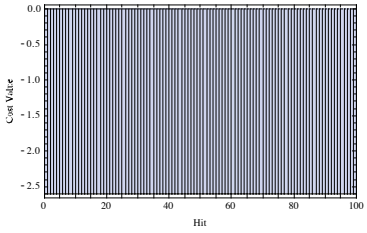
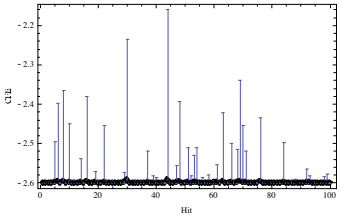
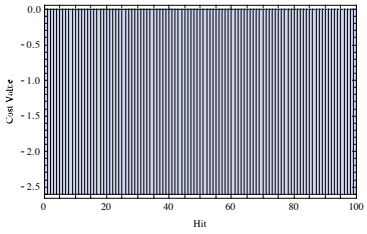
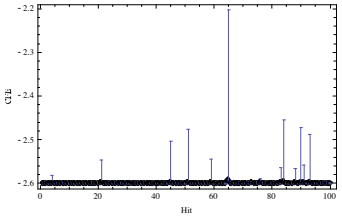




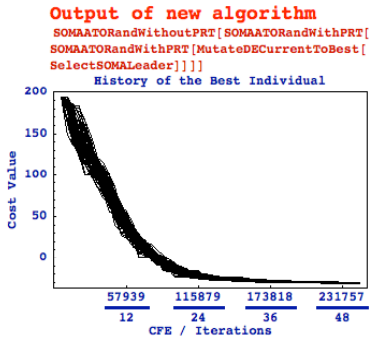


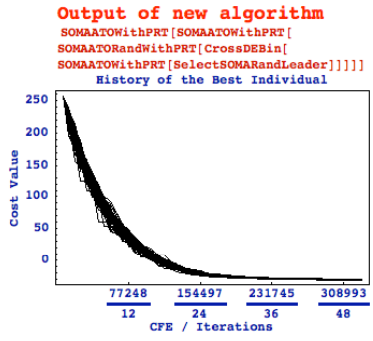


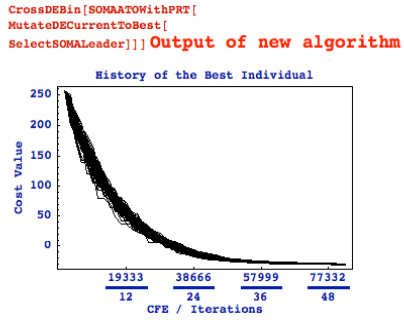


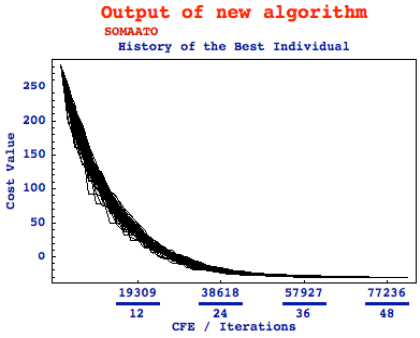
Algorithm 1	
	
Algorithm 2	
	
Algorithm 3	
	
SOMAATO	
	

7.29. Test function (Ackley) – 20D

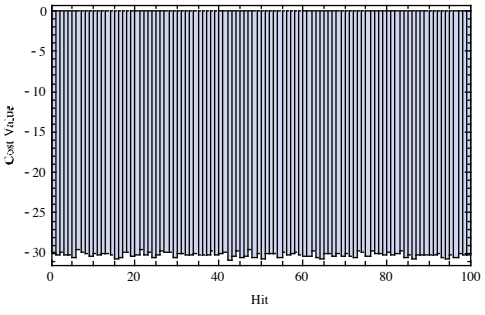


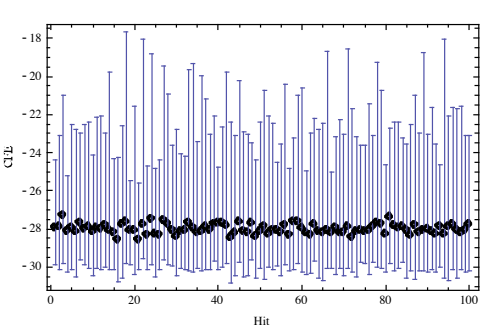




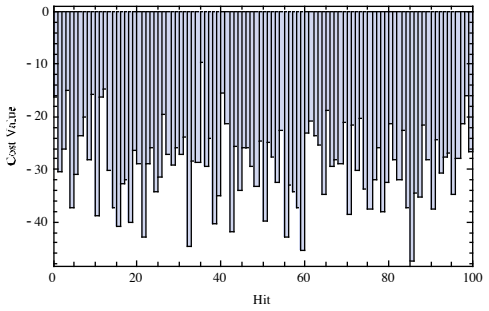


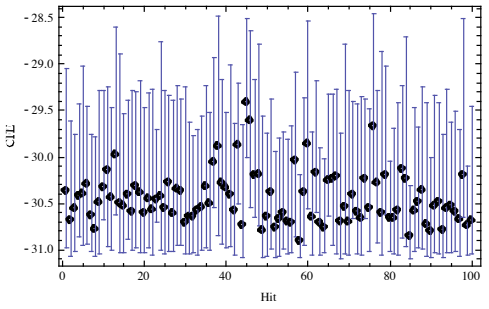
Algorithm 1



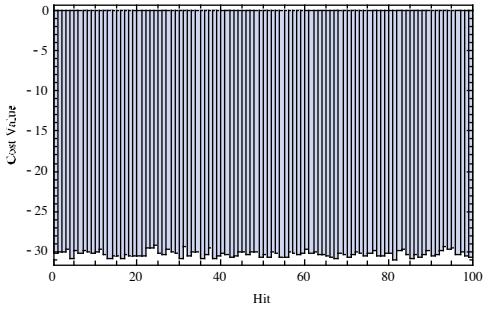


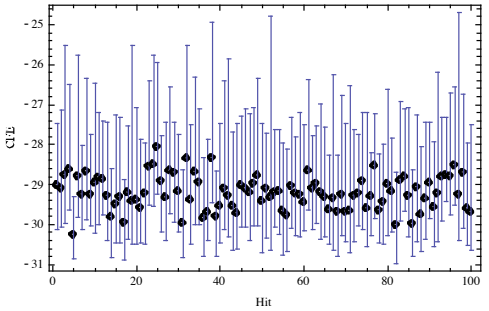
Algorithm 2



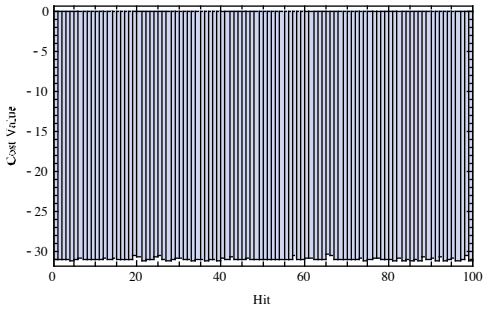


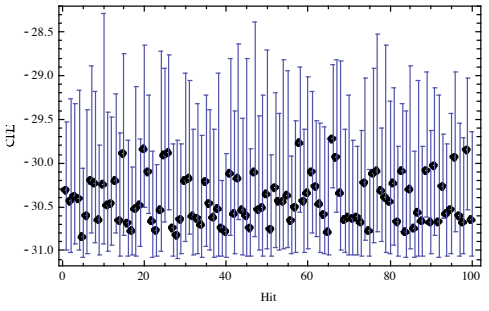
Algorithm 3



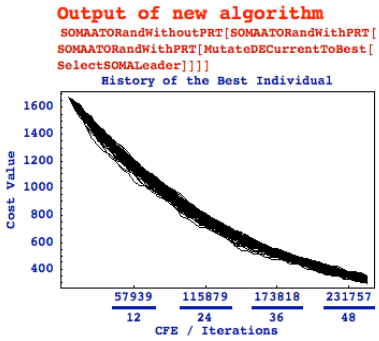


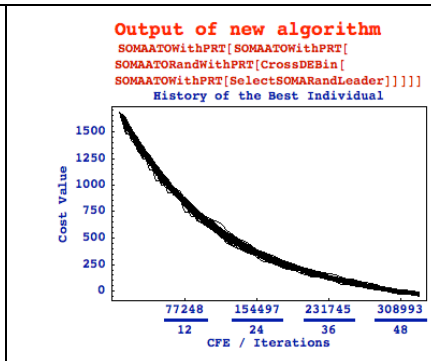
SOMAATO

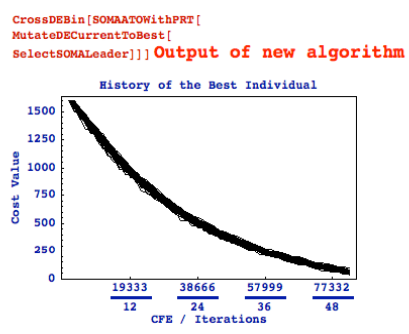


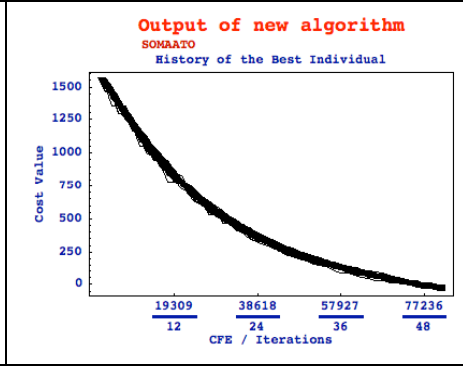


7.30. Test function (Ackley) – 100D

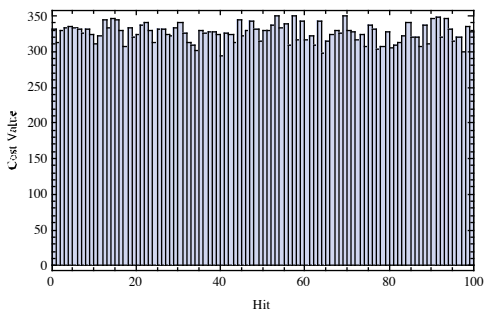


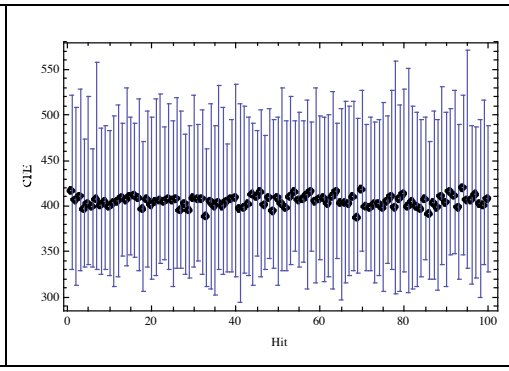




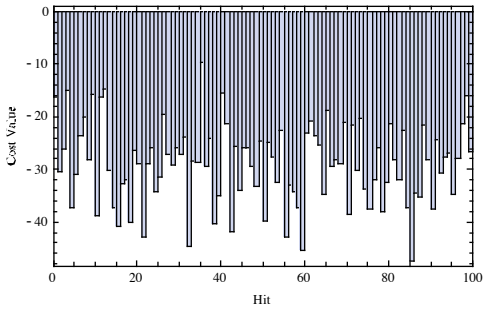


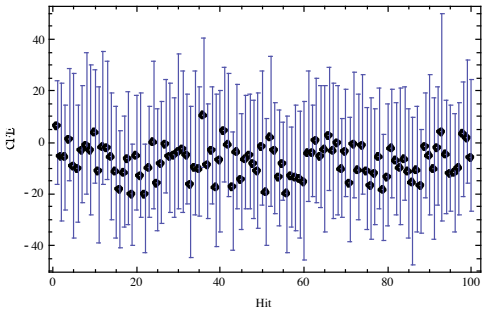
Algorithm 1



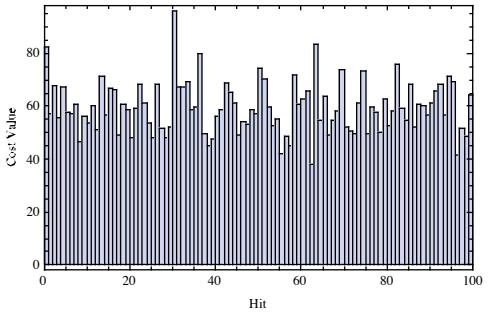


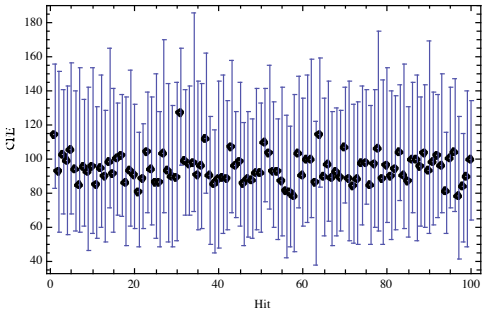
Algorithm 2



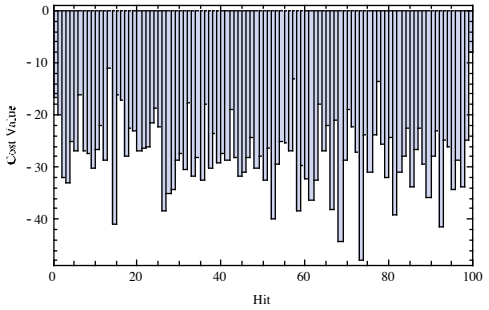


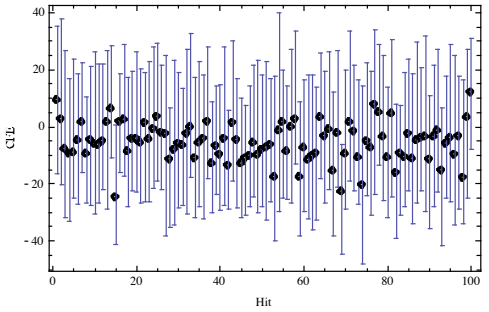
Algorithm 3





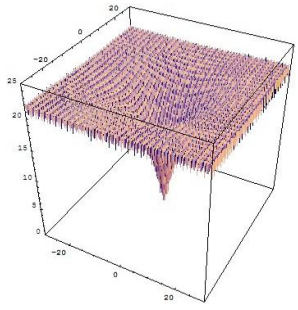
SOMAATO

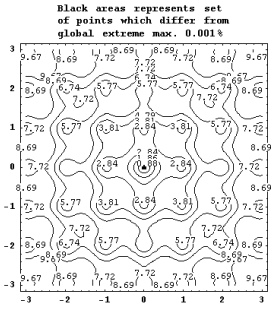


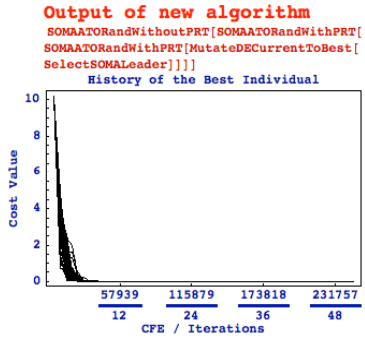


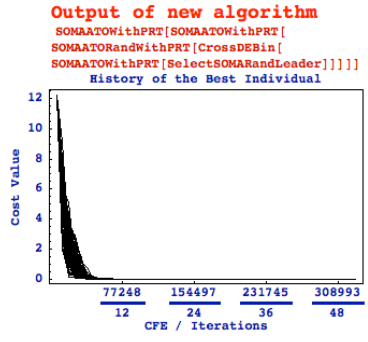
7.31. Ackley‘s function – 2D

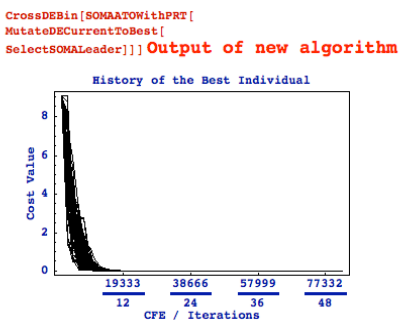
$$f(x) = \sum_{i=1}^{Dim-1} \left(20 + e^{-20} e^{-0.2\sqrt{0.5\left(x_{i+1}^{\ 2} + x_i^{\ 2}\right)}} - e^{0.5(\cos(2\pi x_{i+1}) + \cos(2\pi x_i))} \right)$$

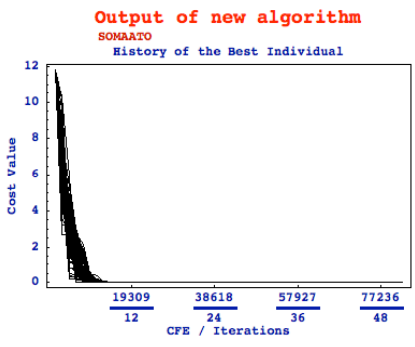




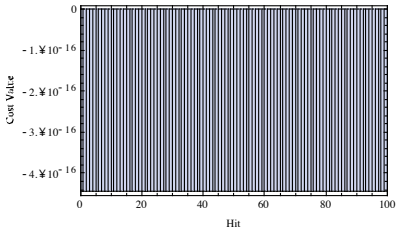


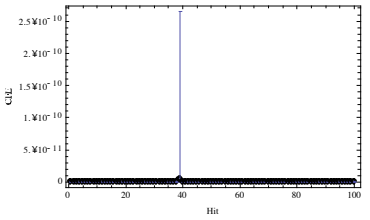




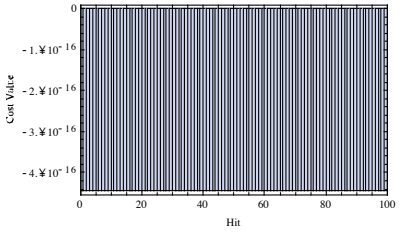


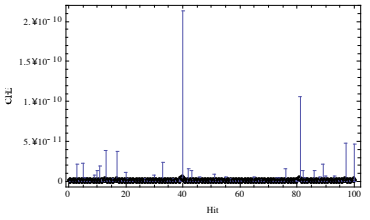
Algorithm 1



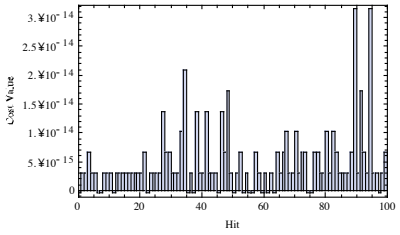


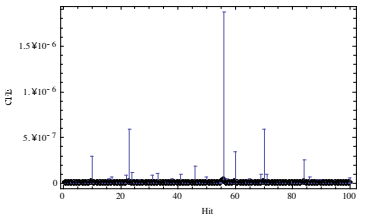
Algorithm 2



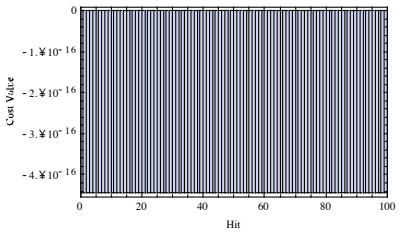


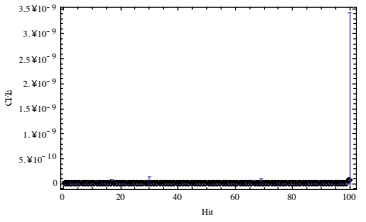
Algorithm 3



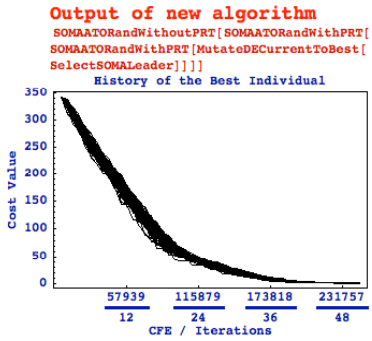


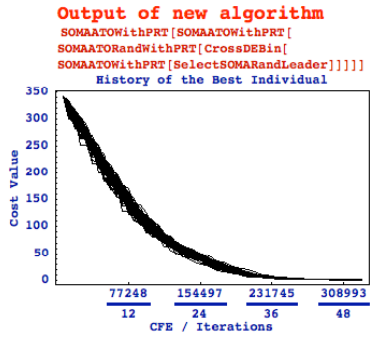
SOMAATO

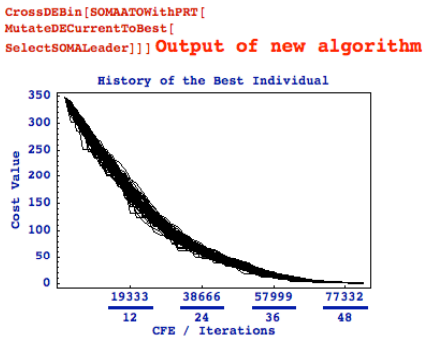


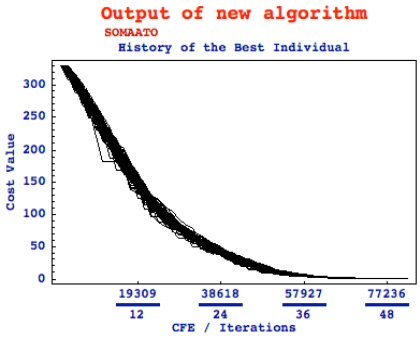


7.32. Ackley‘s function – 20D

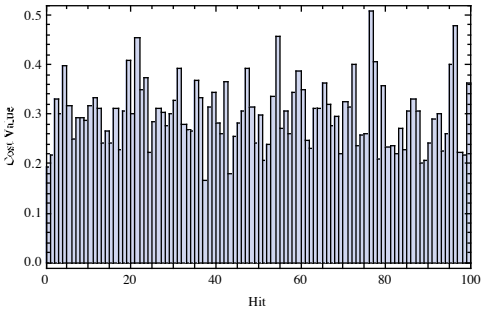


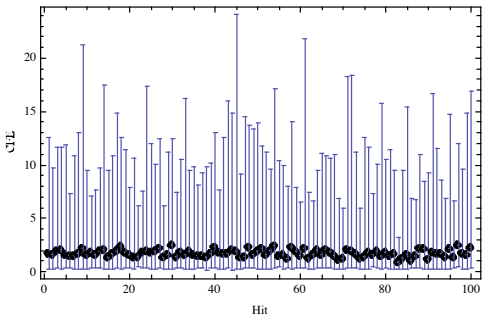




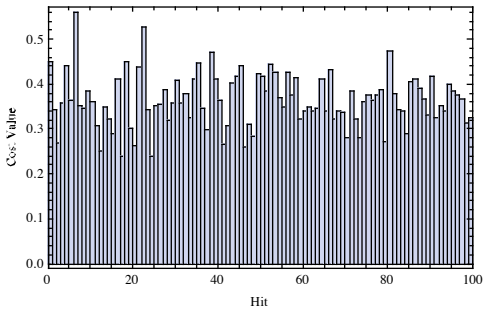


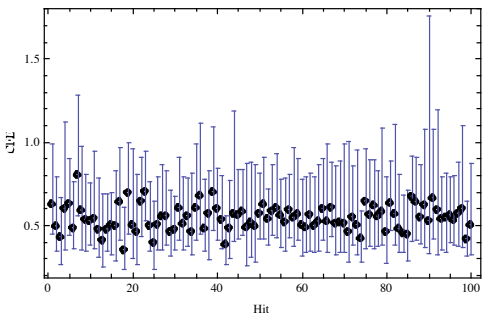
Algorithm 1



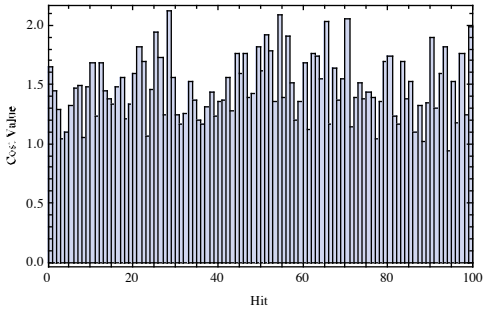


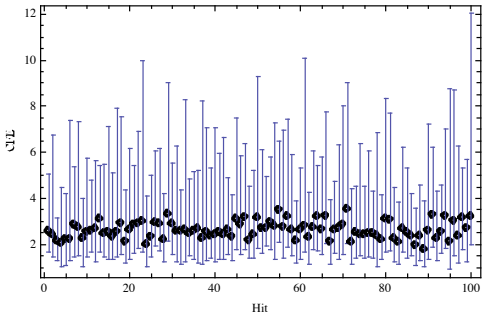
Algorithm 2



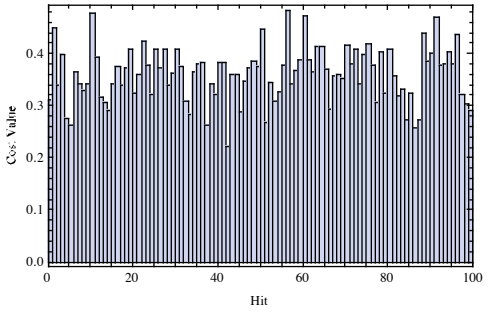


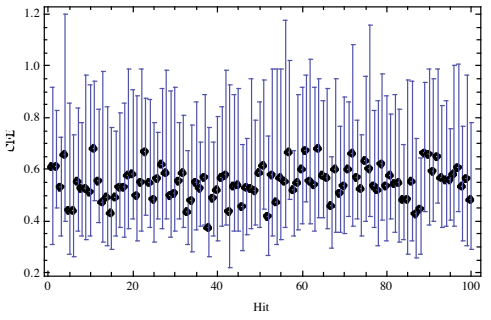
Algorithm 3



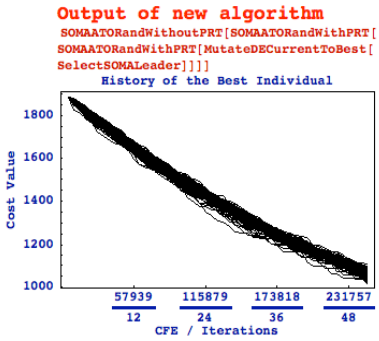


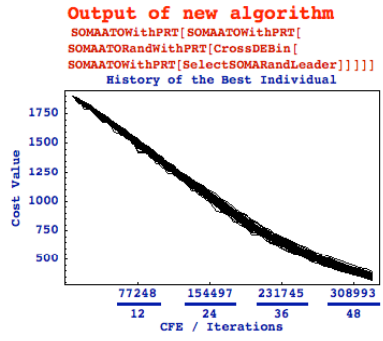
SOMAATO

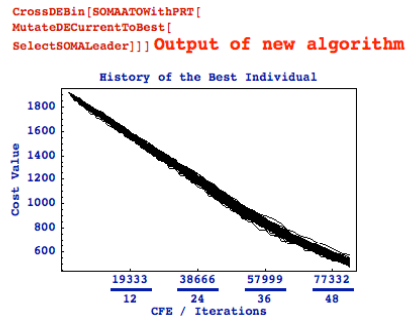


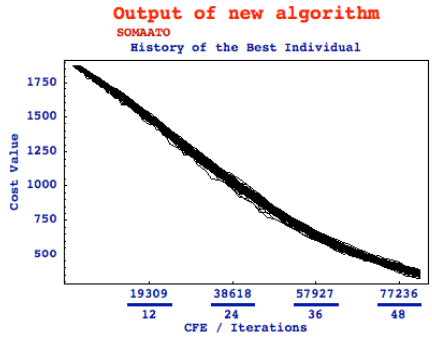


7.33. Ackley‘s function – 100D

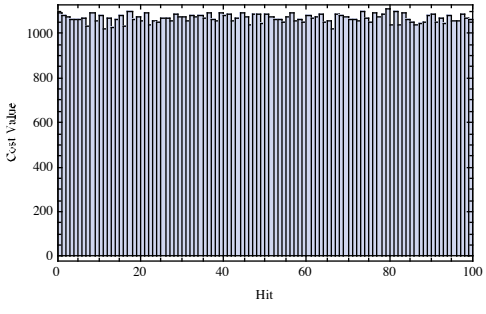


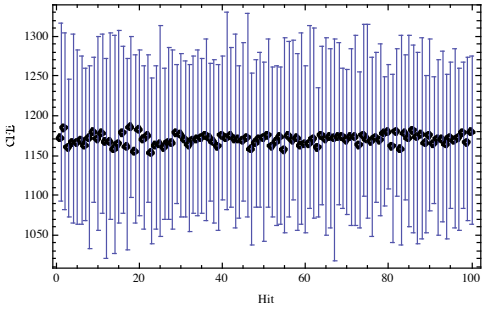




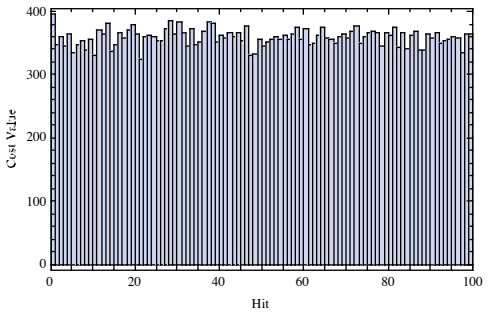


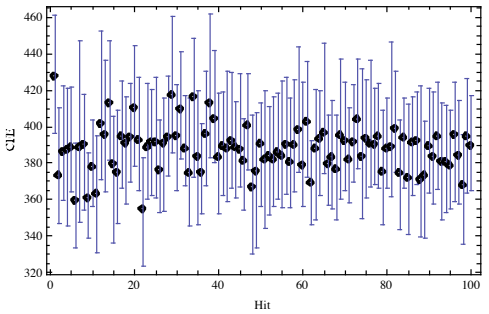
Algorithm 1



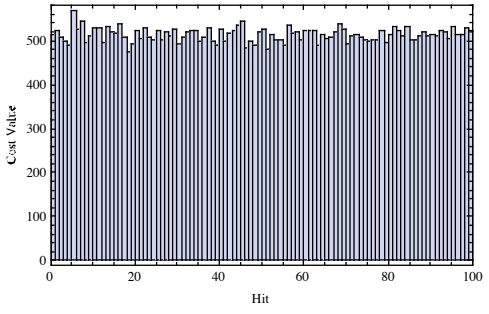


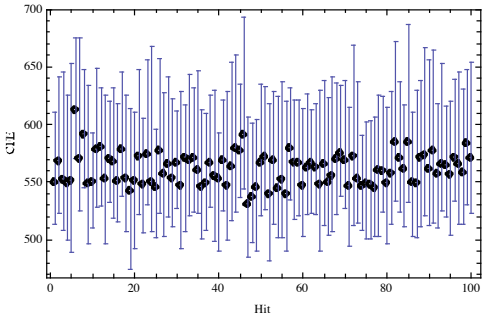
Algorithm 2



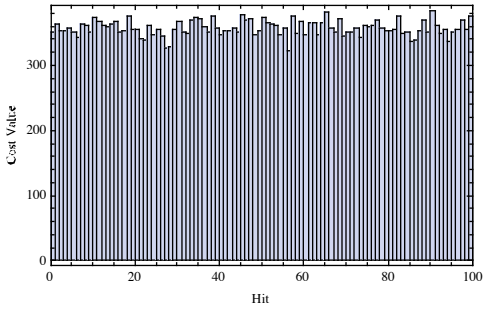


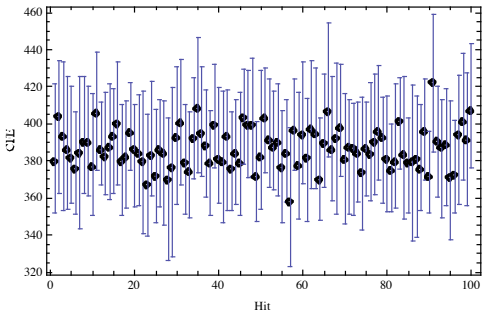
Algorithm 3





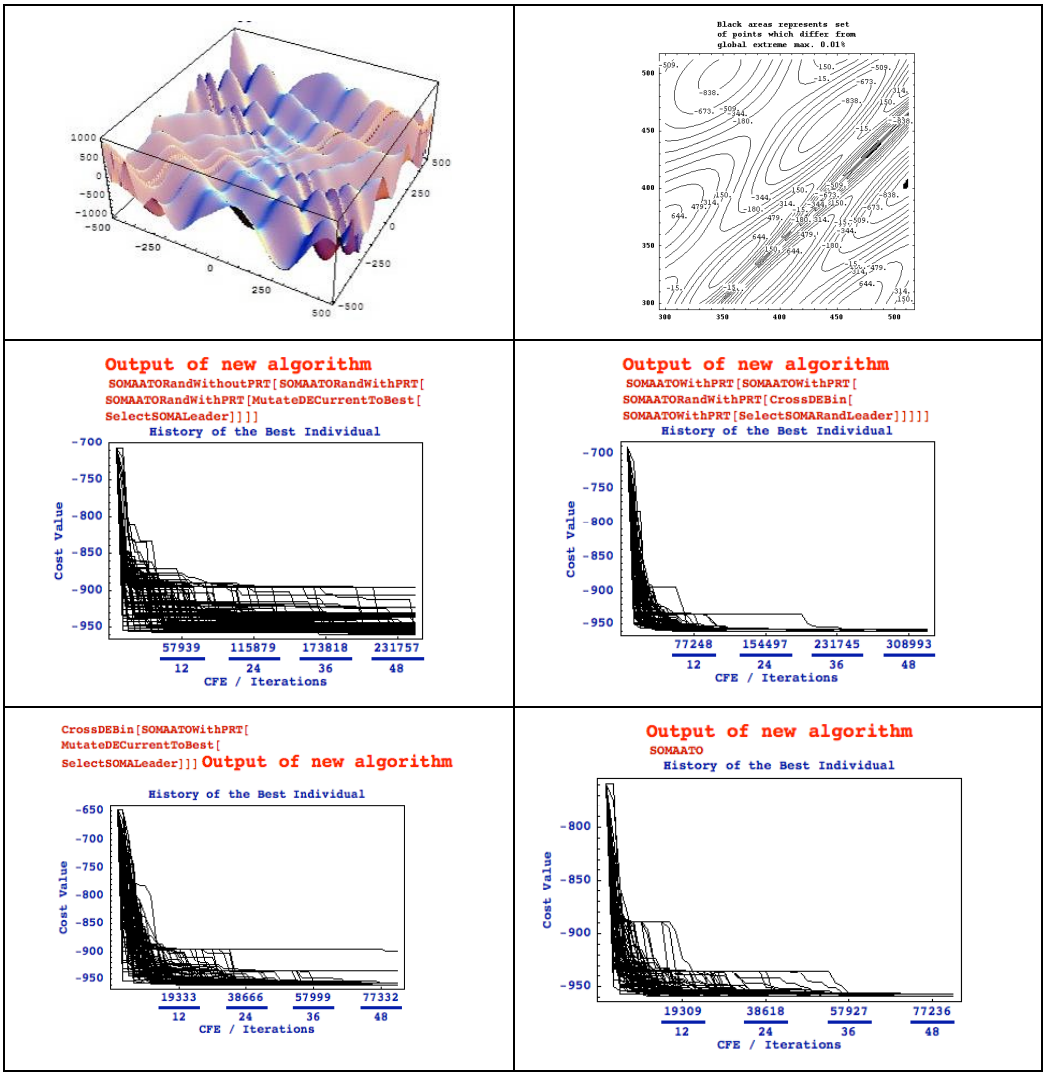
SOMAATO

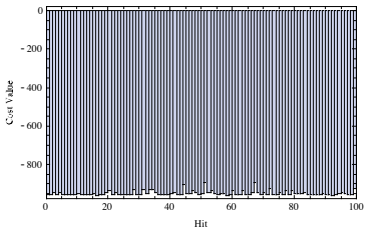
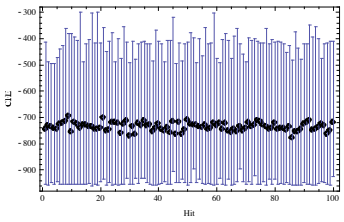
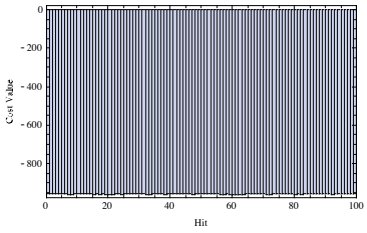
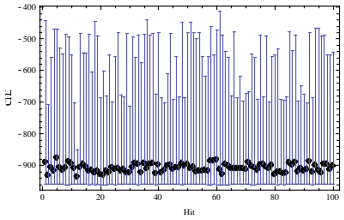
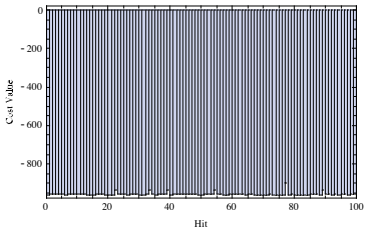
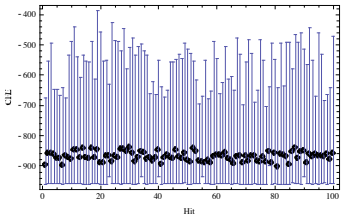
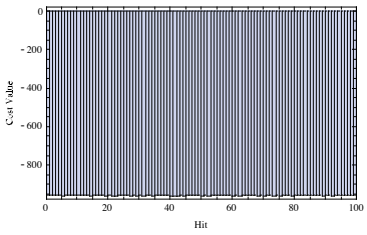
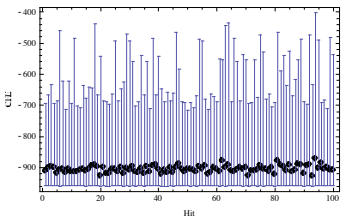




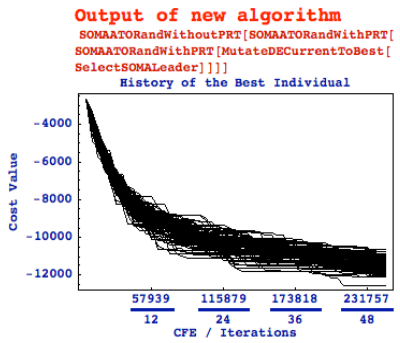
7.34. Egg Holder – 2D

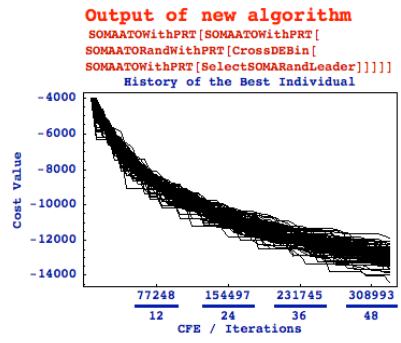
$$f(x) = \sum_{i=1}^{Dim-1} \left(-(x_{i+1} + 47) \sin\left(\sqrt{\left| x_{i+1} + \frac{x_i}{2} + 47 \right|} \right) + \sin\left(\sqrt{\left| x_i - (x_{i+1} + 47) \right|} \right) (-x_i) \right)$$

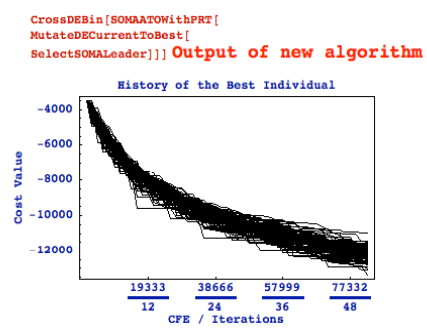


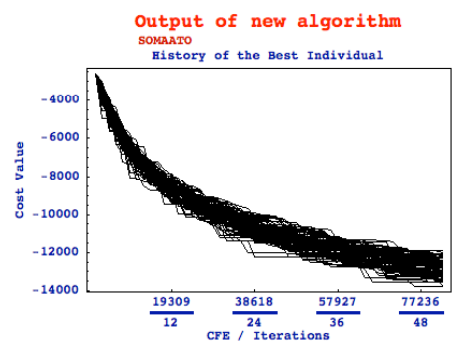
Algorithm 1	
	
Algorithm 2	
	
Algorithm 3	
	
SOMAATO	
	

7.35. Egg Holder – 20D

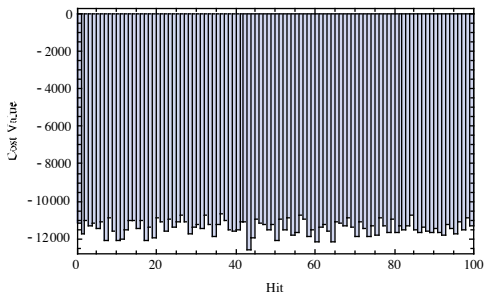


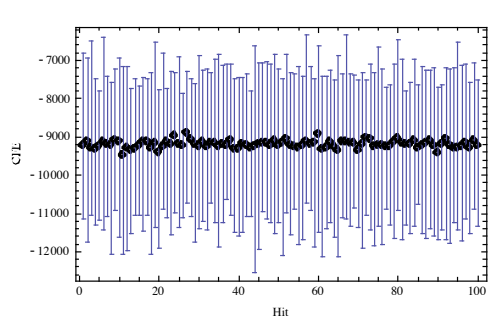




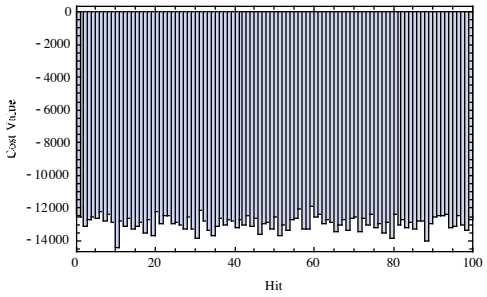


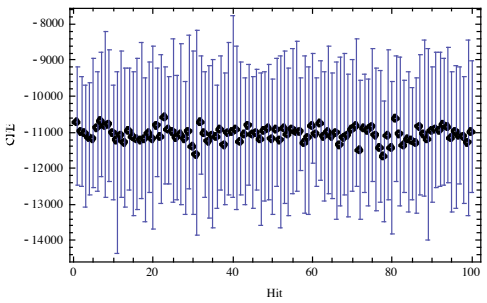
Algorithm 1



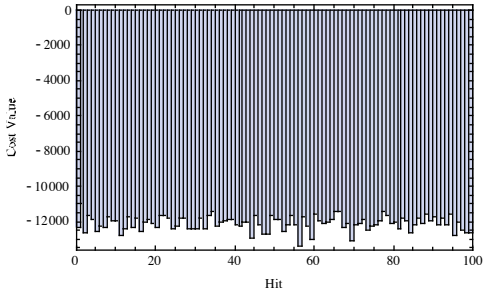


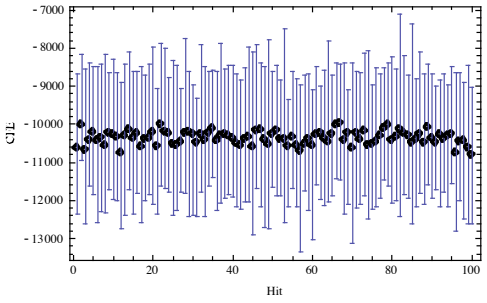
Algorithm 2



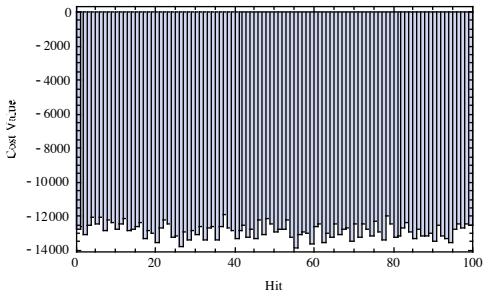


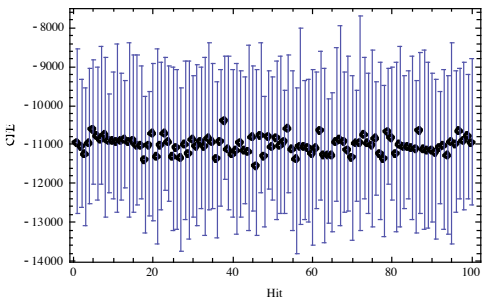
Algorithm 3



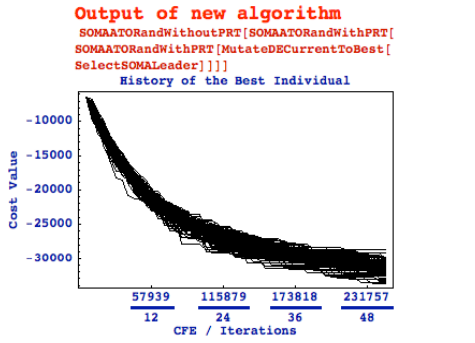


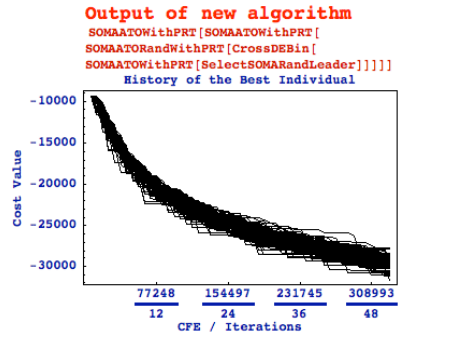
SOMAATO

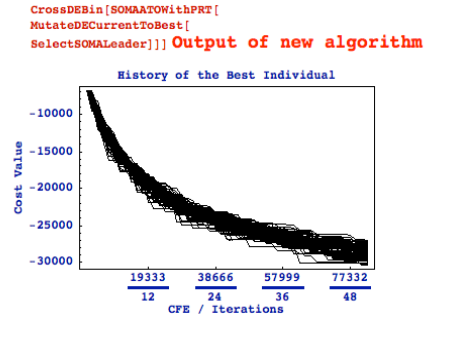


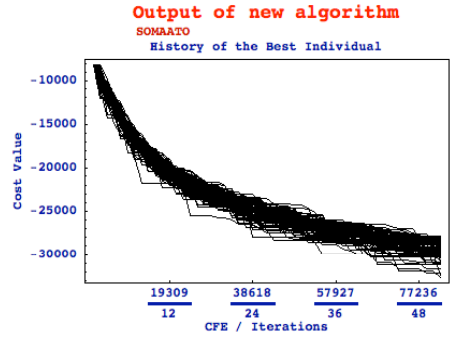


7.36. Egg Holder – 100D

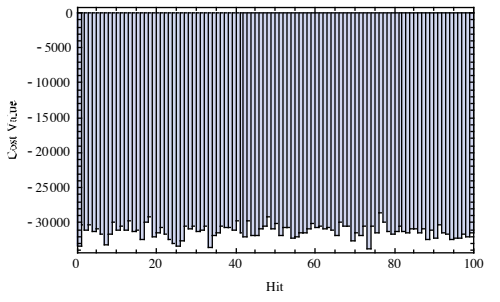


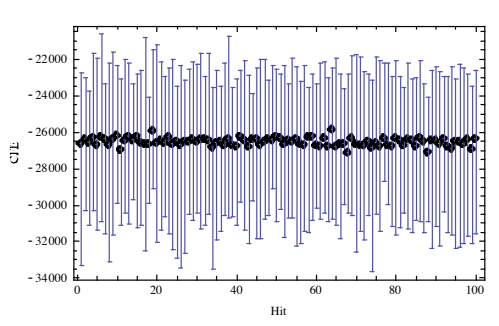




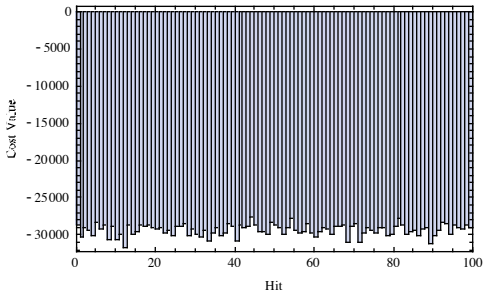


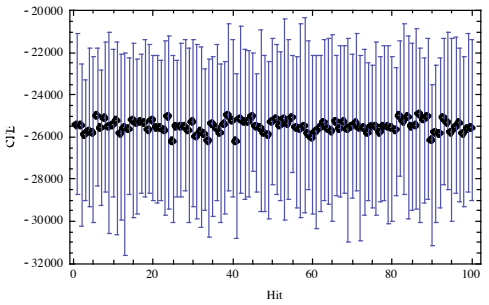
Algorithm 1



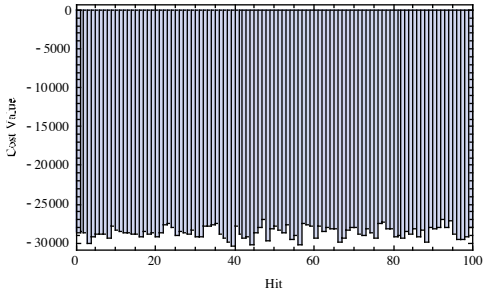


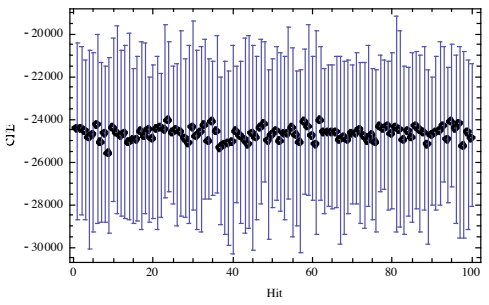
Algorithm 2



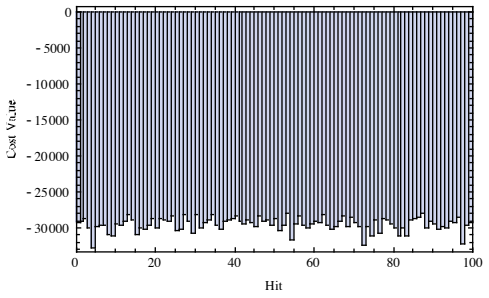


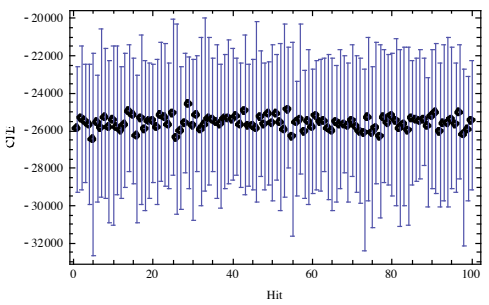
Algorithm 3





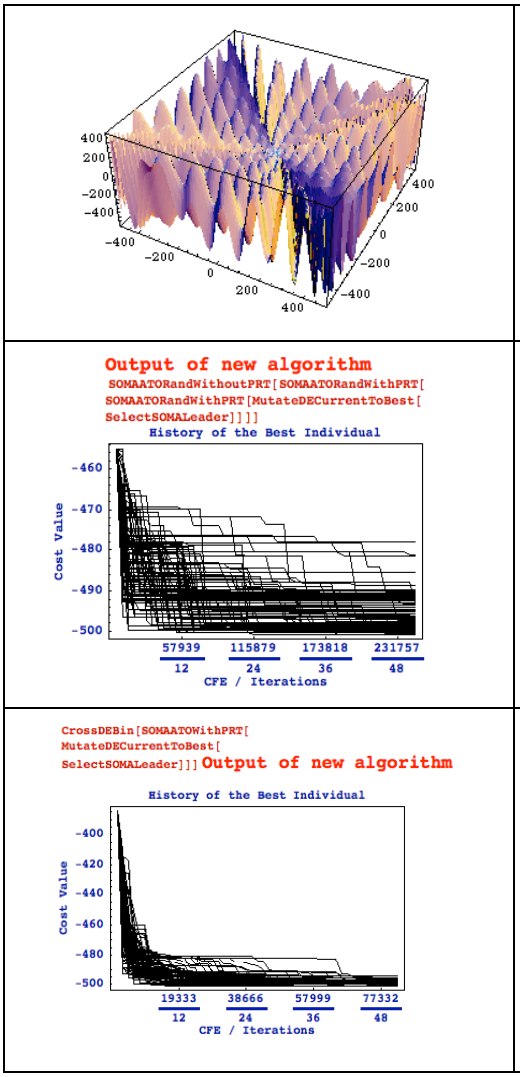
SOMAATO

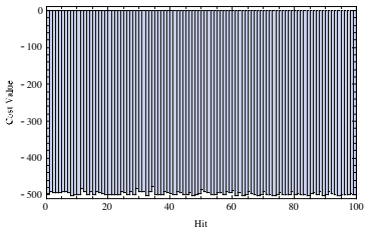
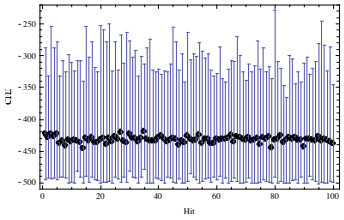
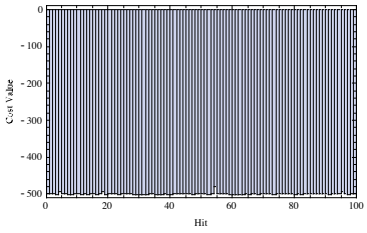
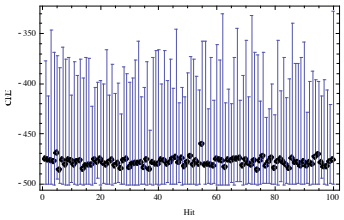
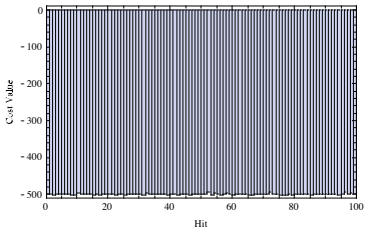
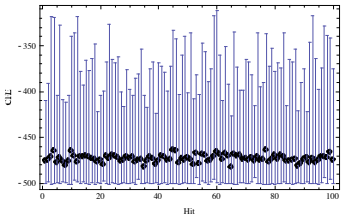
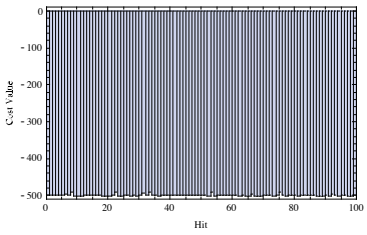
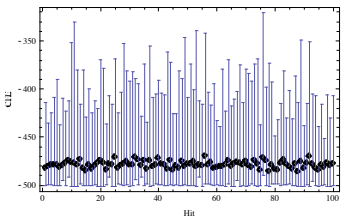




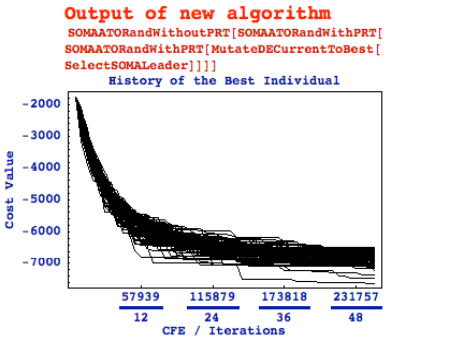
7.37. Rana's function – 2D

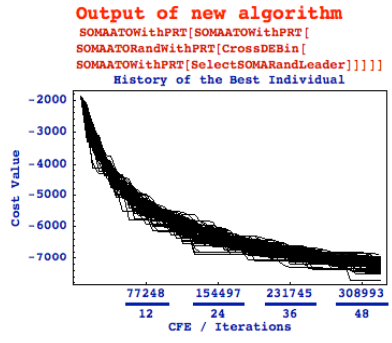
$$f(x) = \sum_{i=1}^{Dim-1} \begin{pmatrix} (x_{i+1}+1)\cos\left(\sqrt{|x_{i+1}-x_i+1|}\right)\sin\left(\sqrt{|x_{i+1}+x_i+1|}\right) \\ +\cos\left(\sqrt{|x_{i+1}+x_i+1|}\right)\sin\left(\sqrt{|x_{i+1}-x_i+1|}\right)x_i \end{pmatrix}$$

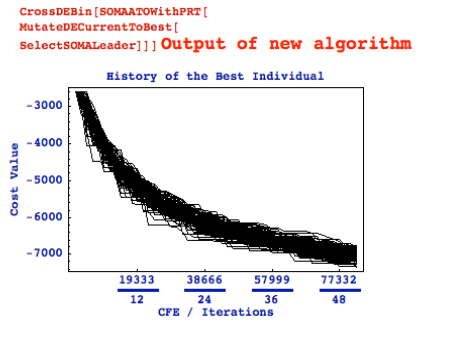


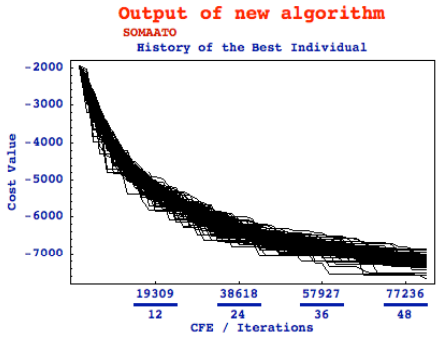
Algorithm 1	
	
Algorithm 2	
	
Algorithm 3	
	
SOMAATO	
	

7.38. Rana's function – 20D

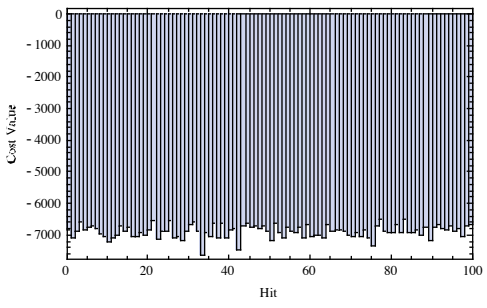


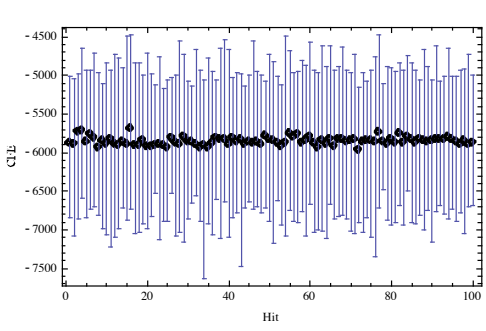




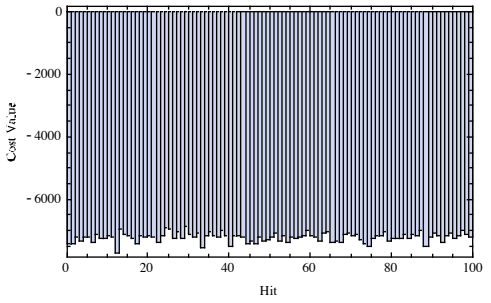


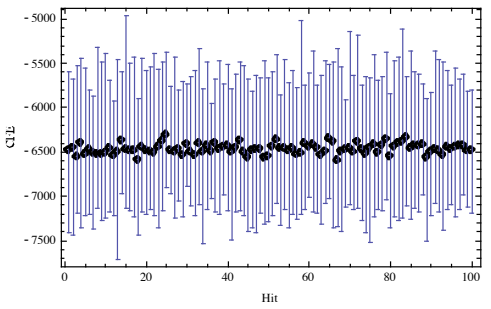
Algorithm 1



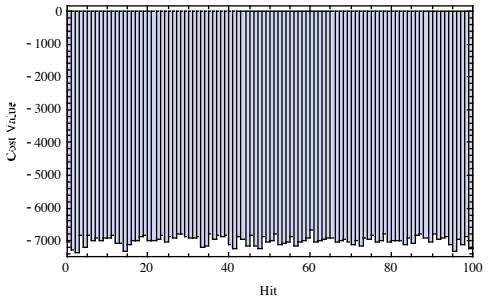


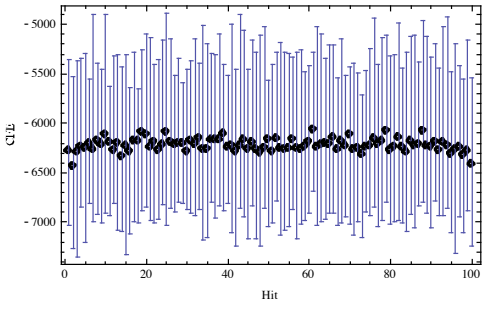
Algorithm 2



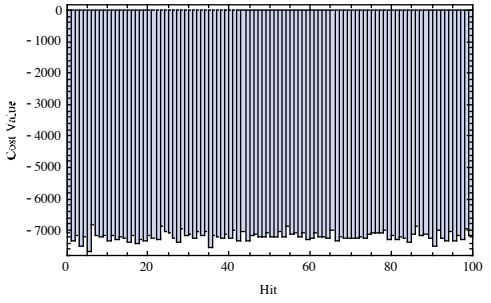


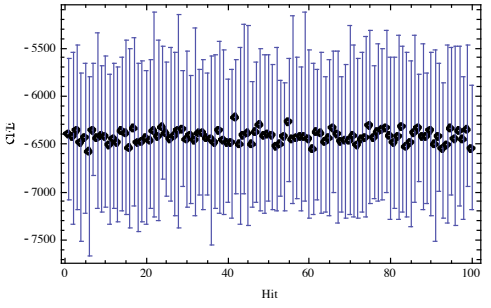
Algorithm 3



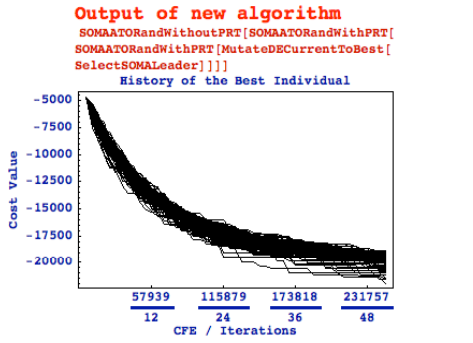


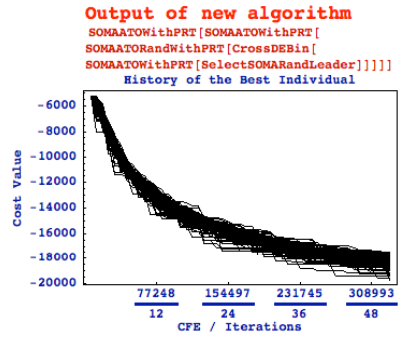
SOMAATO

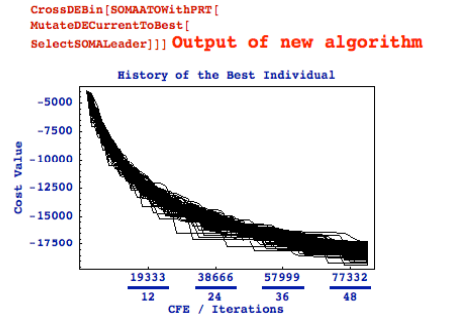


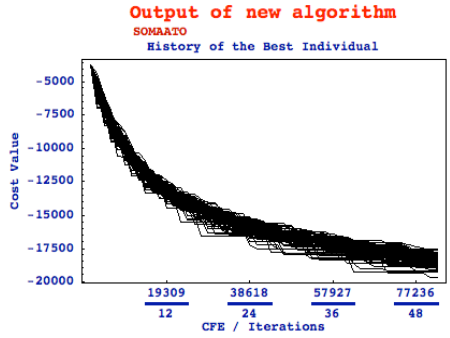


7.39. Rana's function – 100D

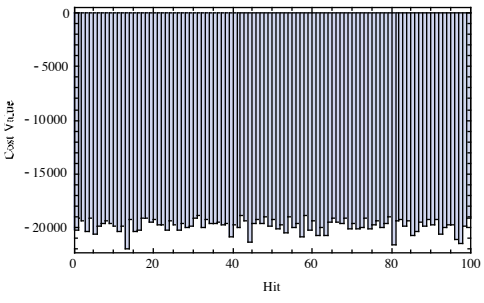


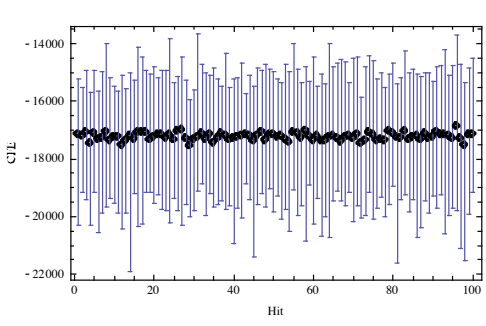




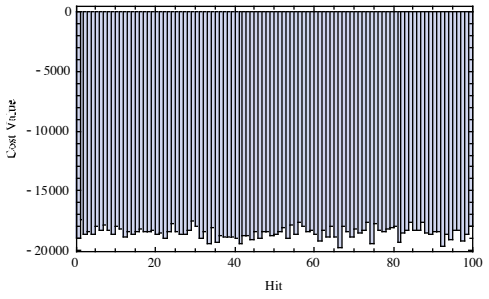


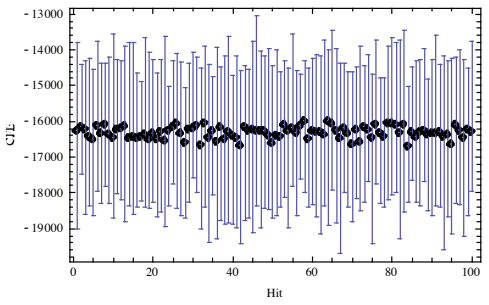
Algorithm 1



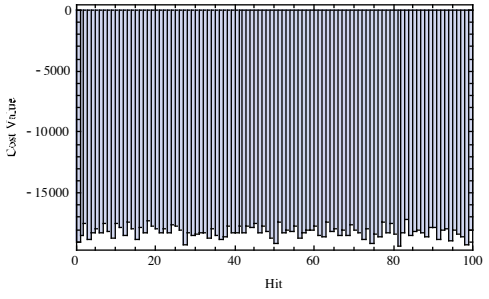


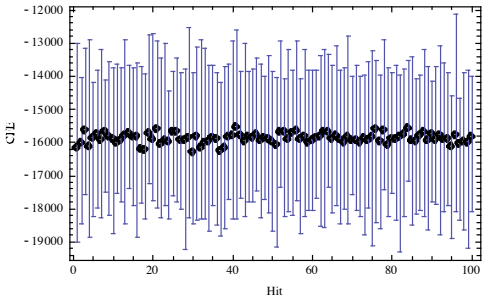
Algorithm 2



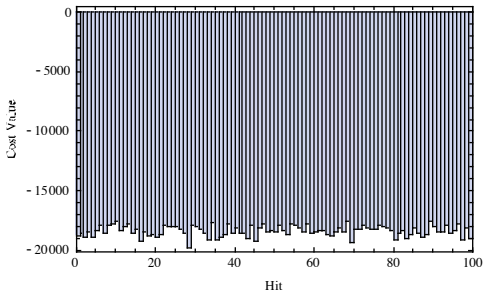


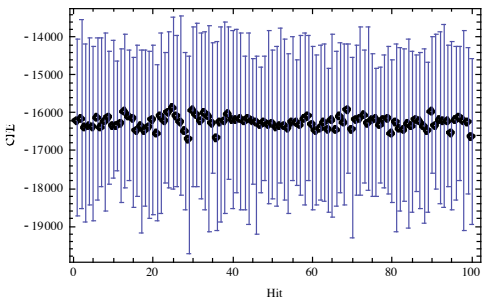
Algorithm 3





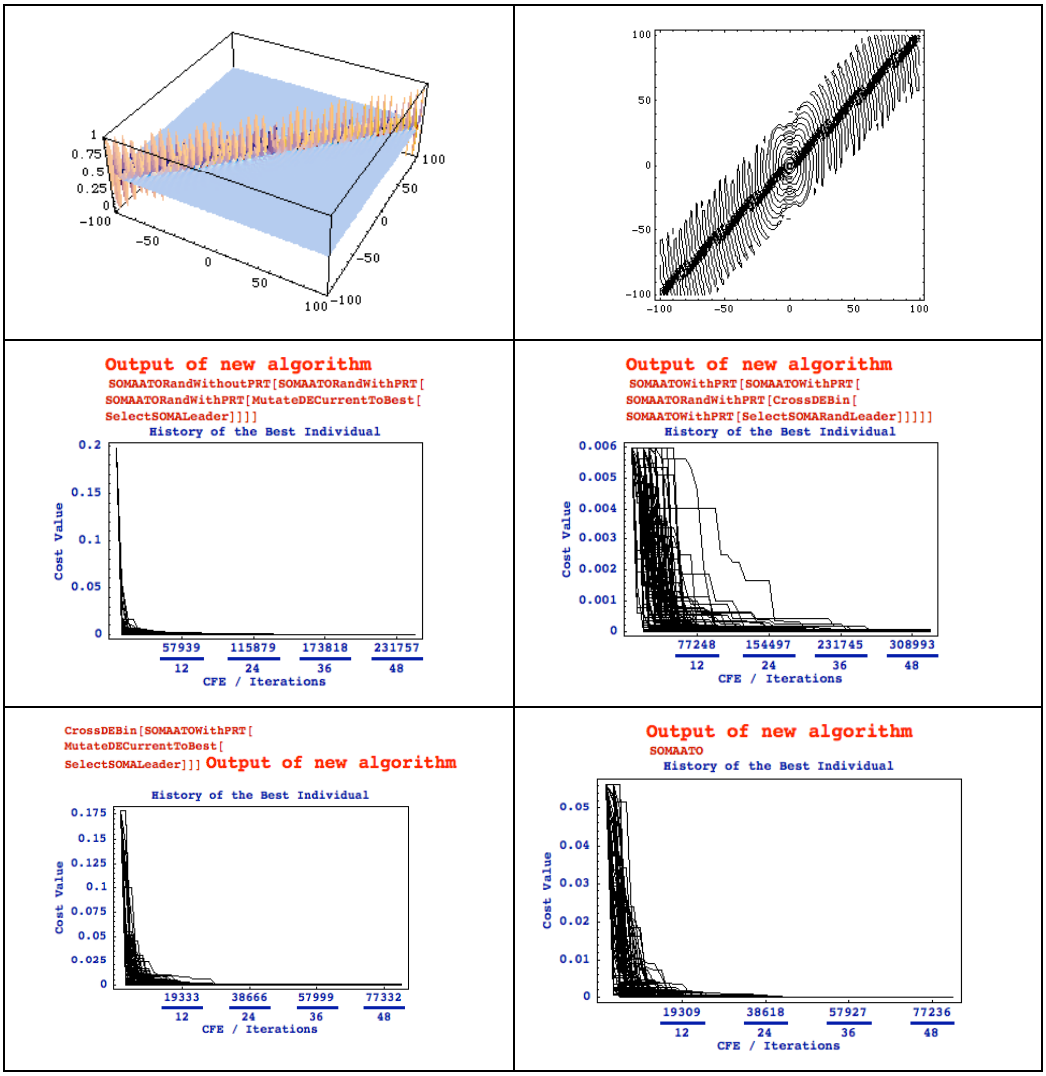
SOMAATO



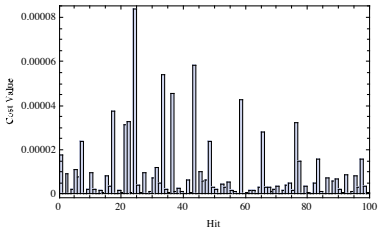


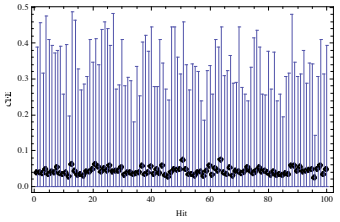
7.40. Pathological test function – 2D

$$f(x) = \sum_{i=1}^{Dim-1} \left(\frac{\sin^2\left(\sqrt{x_{i+1}^2 + 100x_i^2}\right) - 0.5}{0.001\left(x_{i+1}^2 - 2x_i x_{i+1} + x_i^2\right)^2 + 1} + 0.5 \right)$$

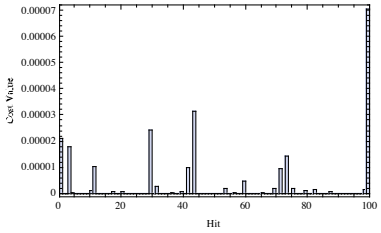


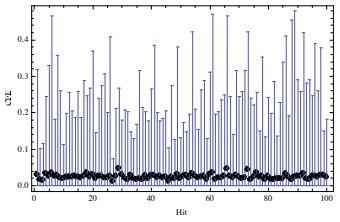
Algorithm 1



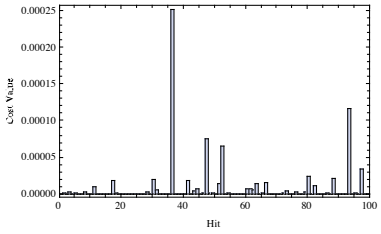


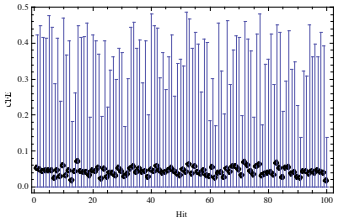
Algorithm 2



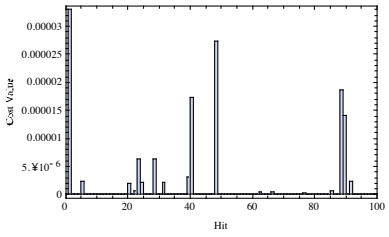


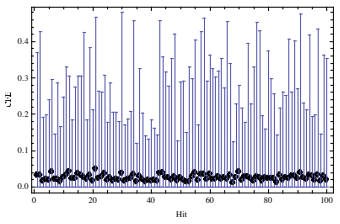
Algorithm 3



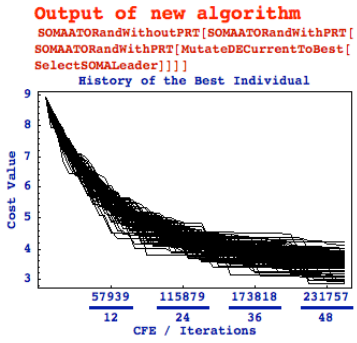


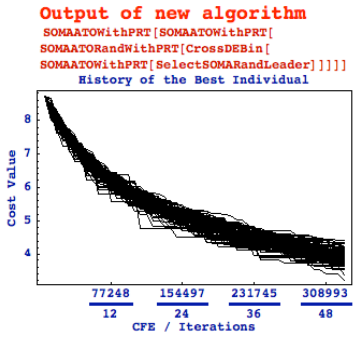
SOMAATO

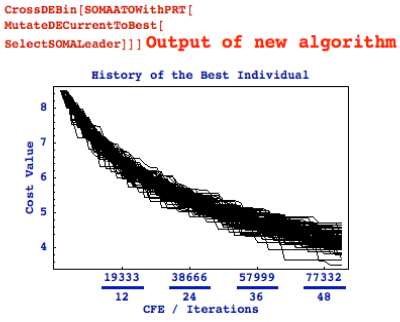


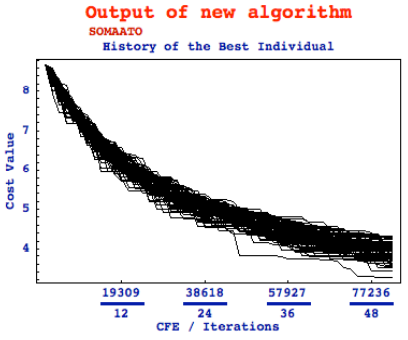


7.41. Pathological test function – 20D

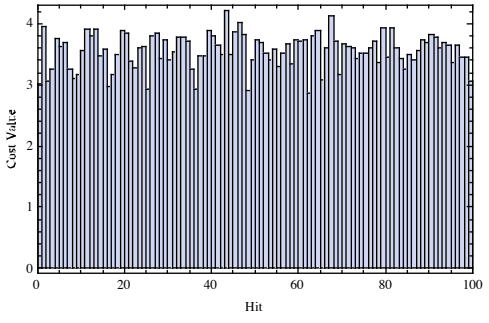


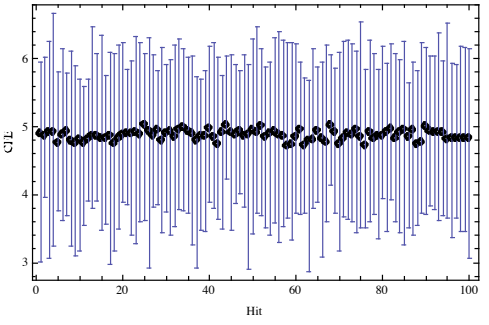




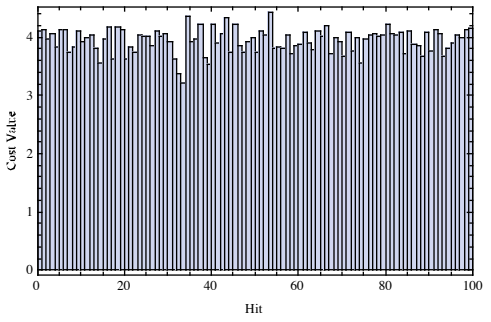


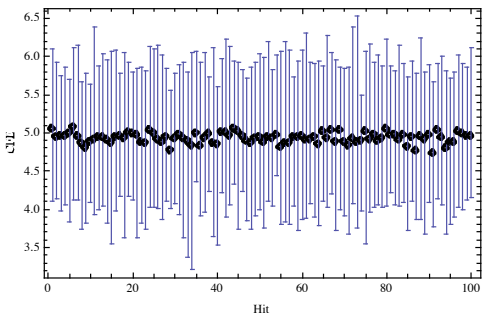
Algorithm 1



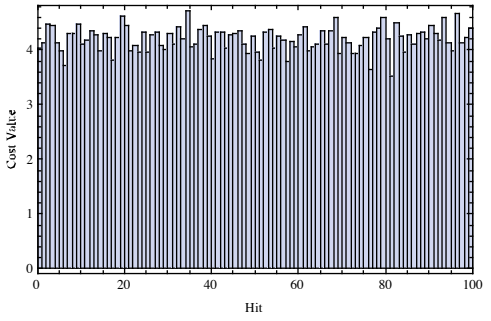


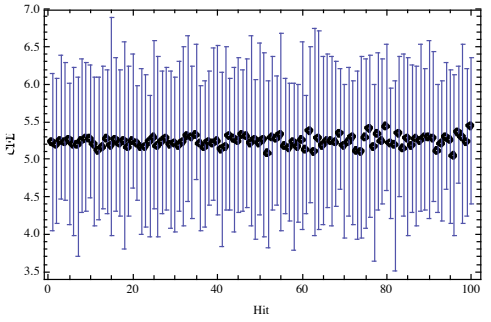
Algorithm 2



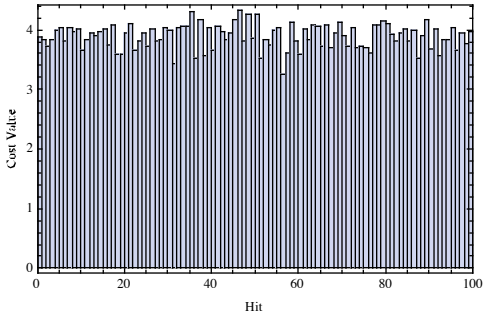


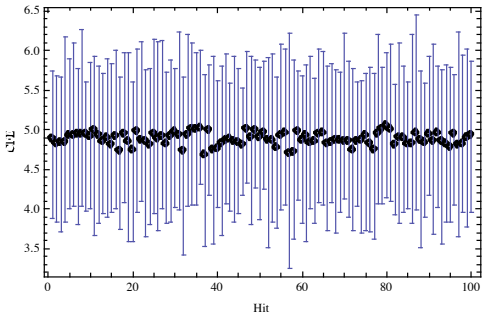
Algorithm 3



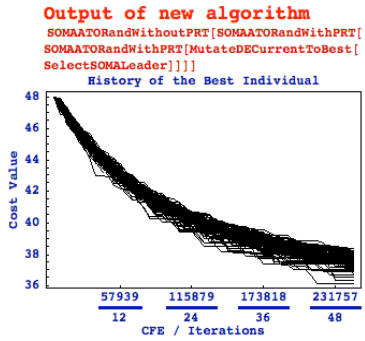


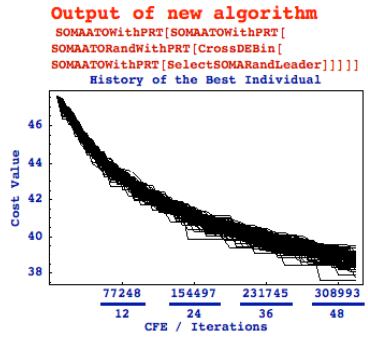
SOMAATO

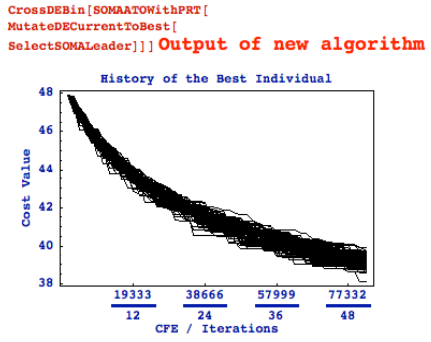


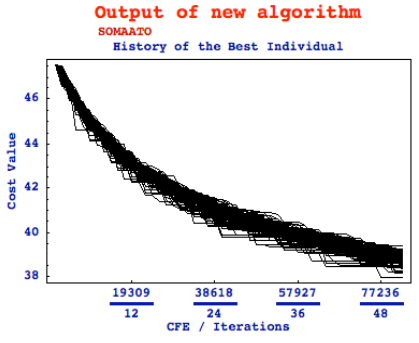


7.42. Pathological test function – 100D

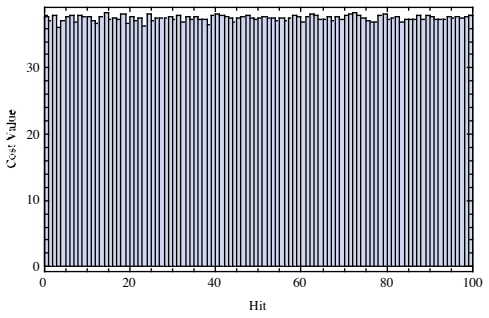


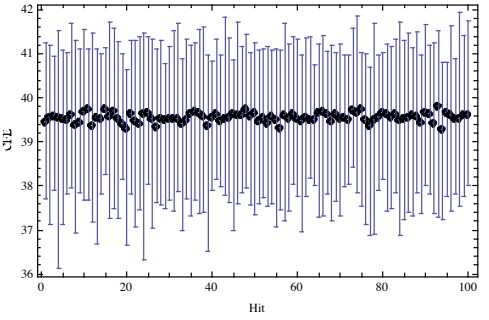




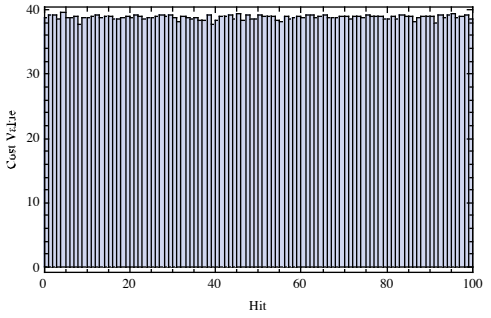


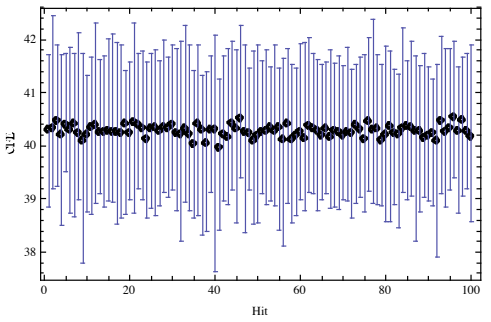
Algorithm 1



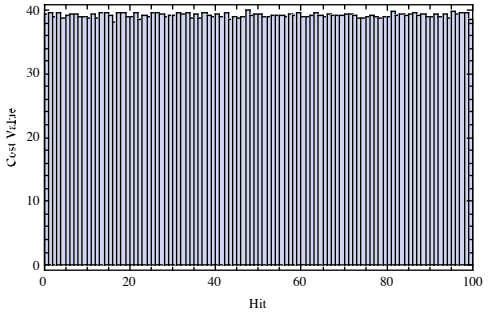


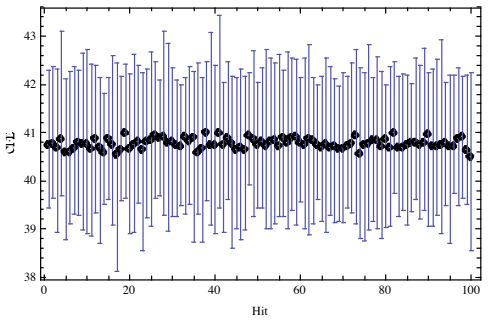
Algorithm 2



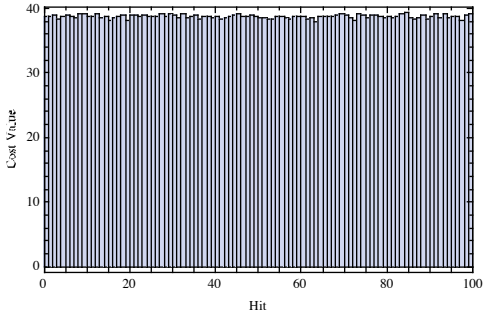


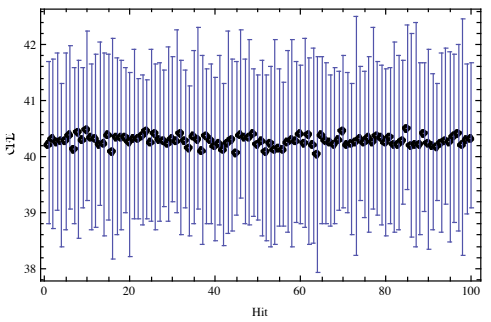
Algorithm 3





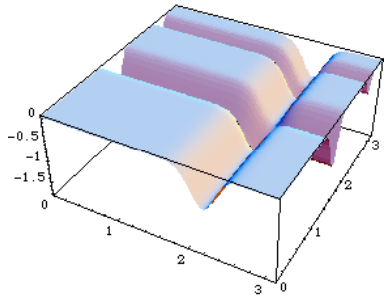
SOMAATO

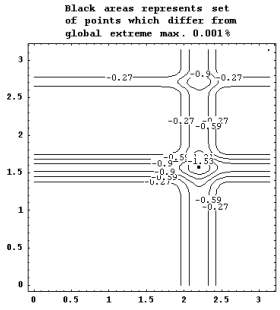


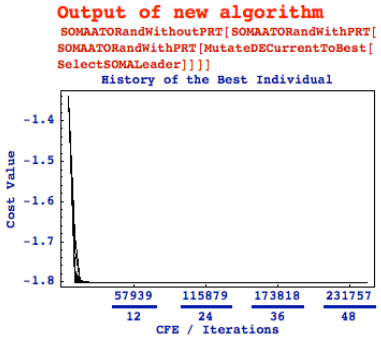


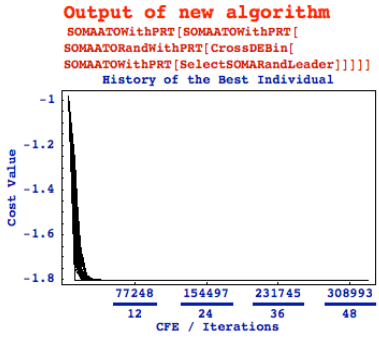
7.43. Michalewicz's function – 2D

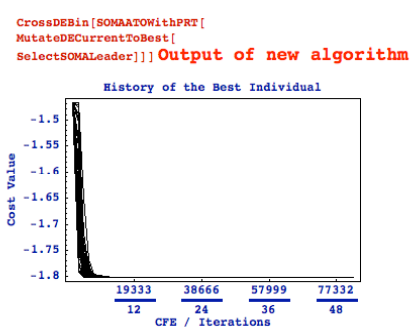
$$f(x) = \sum_{i=1}^{Dim-1} -\left(\sin(x_{i+1})\sin^{20}\left(\frac{2x_{i+1}^2}{\pi}\right) + \sin^{20}\left(\frac{x_i^2}{\pi}\right)\sin(x_i)\right)$$

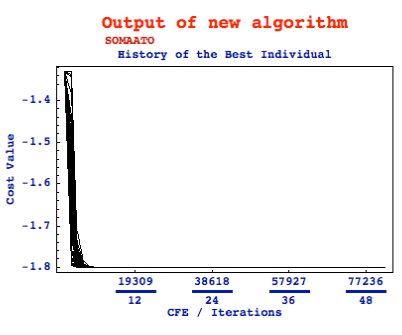


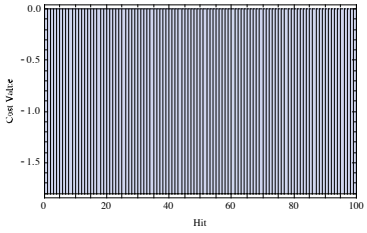
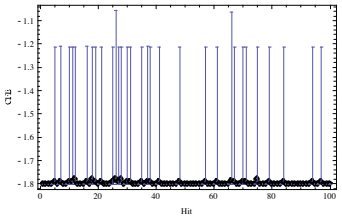
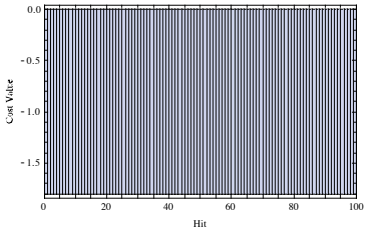
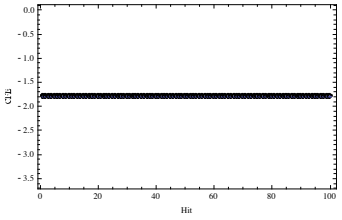
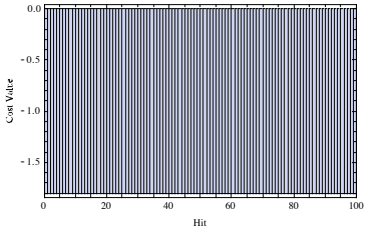
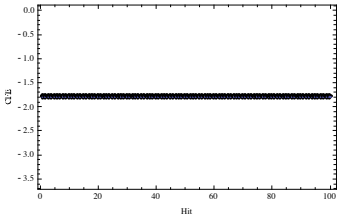
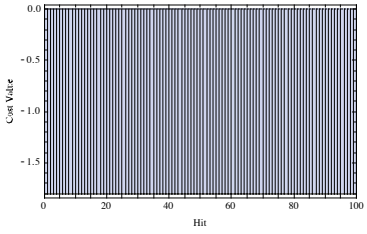
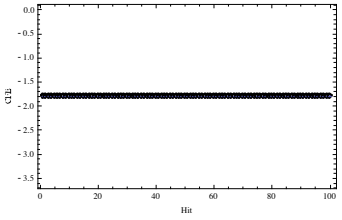




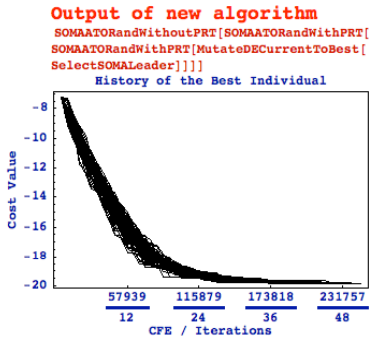


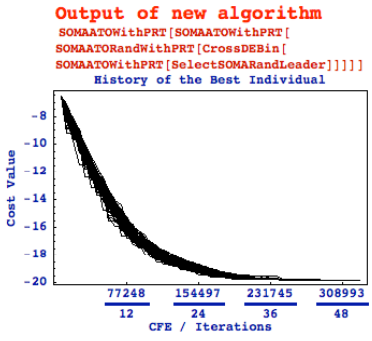


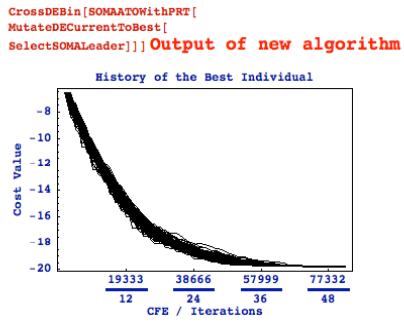


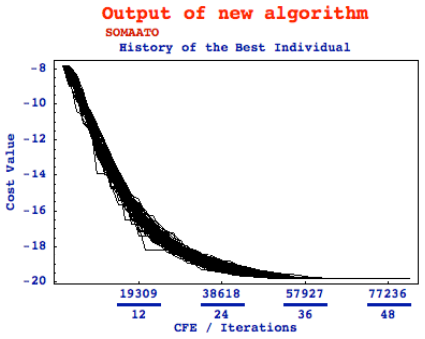
Algorithm 1	
	
Algorithm 2	
	
Algorithm 3	
	
SOMAATO	
	

7.44. Michalewicz‘s function – 20D

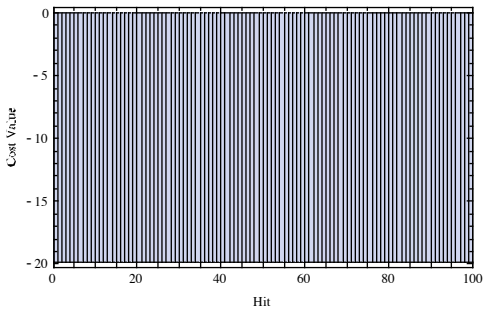


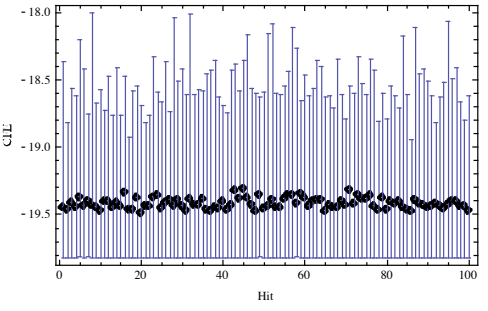




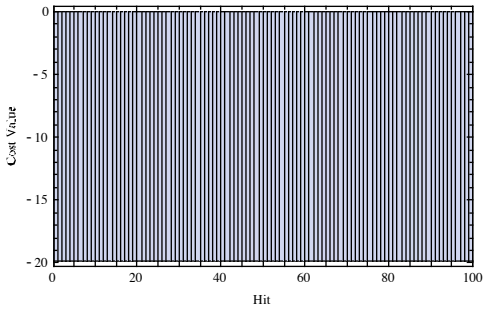


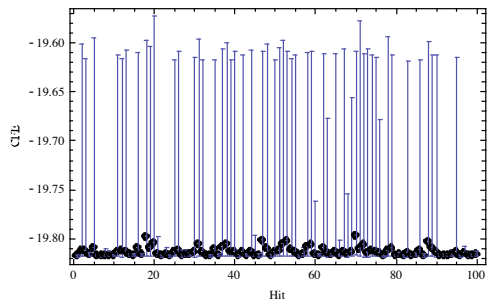
Algorithm 1



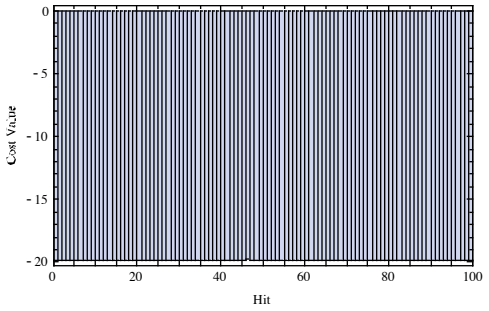


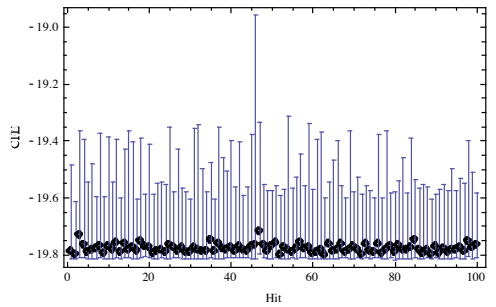
Algorithm 2



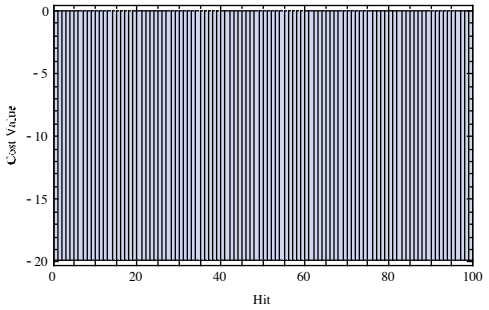


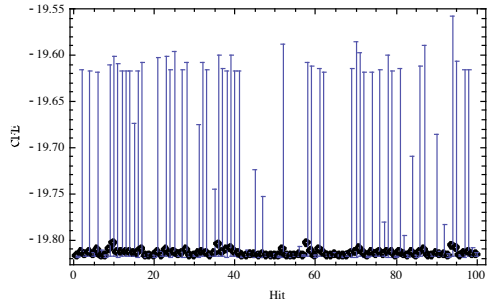
Algorithm 3



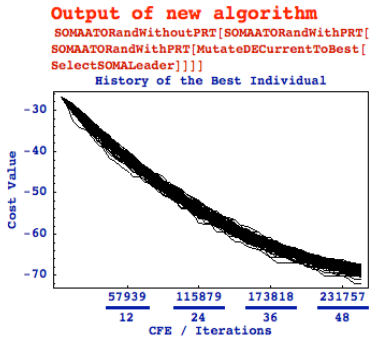


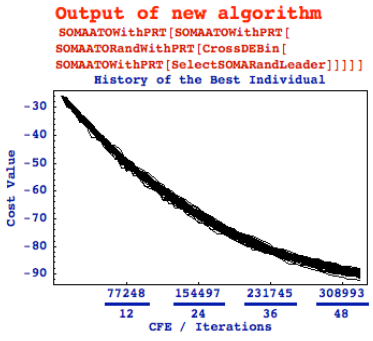
SOMAATO

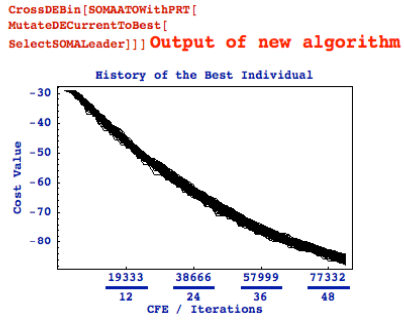


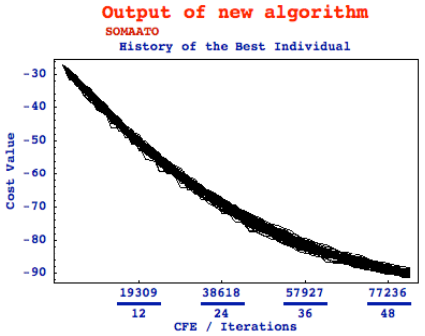


7.45. Michalewicz‘s function – 100D

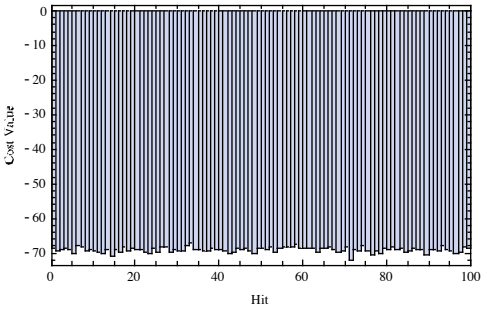


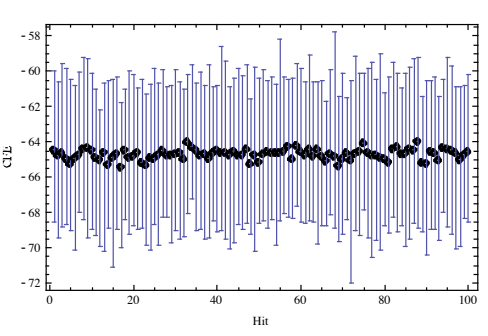




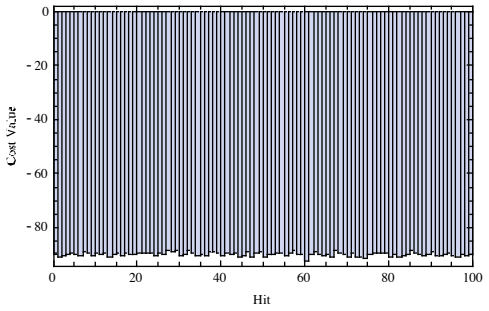


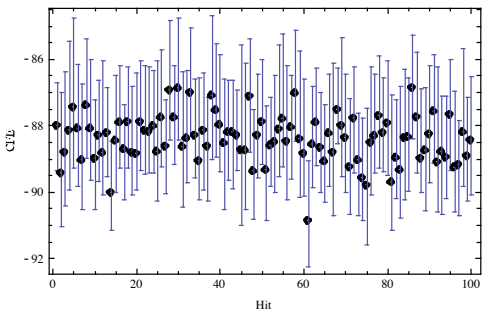
Algorithm 1



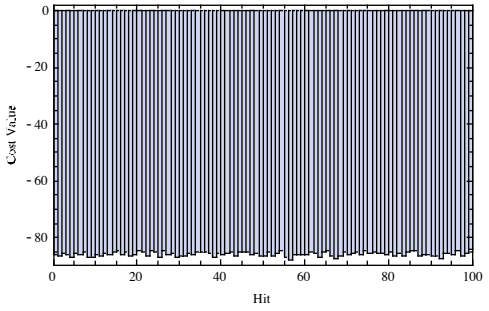


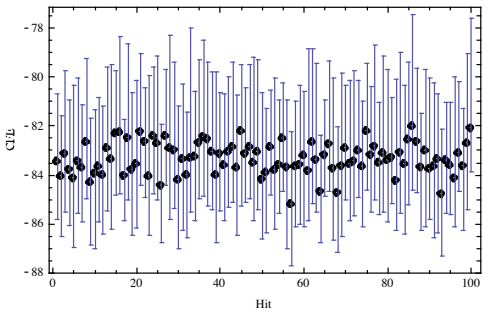
Algorithm 2



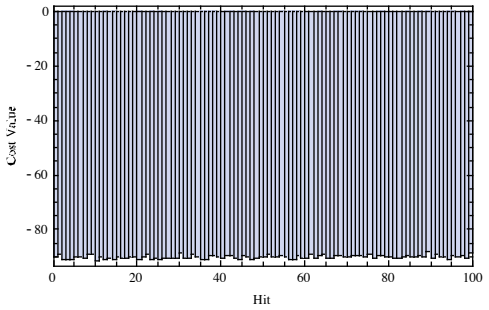


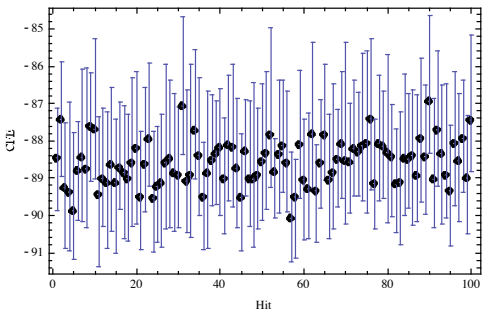
Algorithm 3





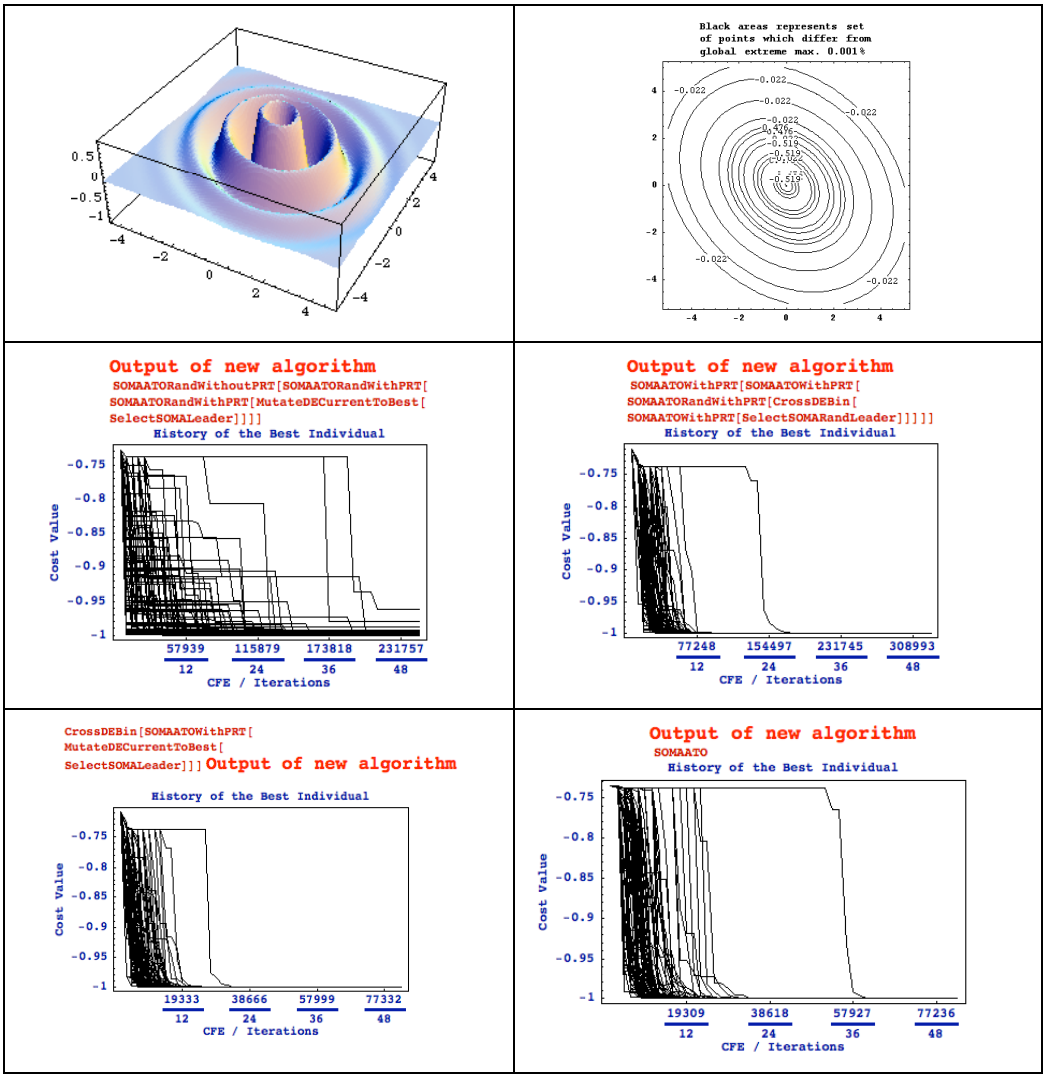
SOMAATO



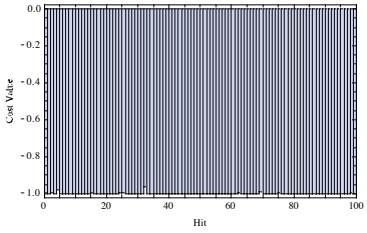


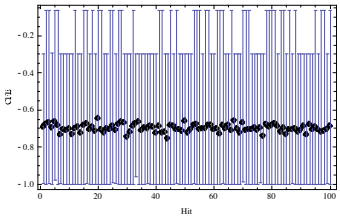
7.46. Master's cosine wave function – 2D

$$f(x) = -\sum_{i=1}^{Dim-1} e^{-\frac{1}{8}\left(x_{i+1}^2 + 0.5x_i x_{i+1} + x_i^2\right)} \cos\left(4\sqrt{x_{i+1}^2 + 0.5x_i x_{i+1} + x_i^2}\right)$$

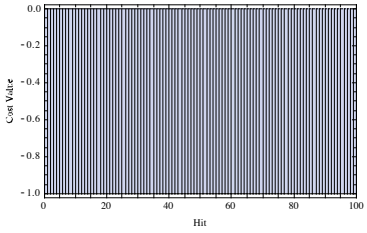


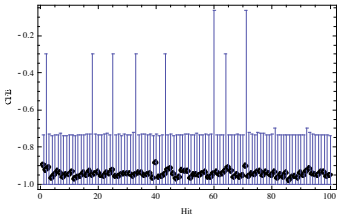
Algorithm 1



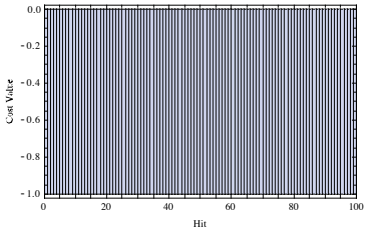


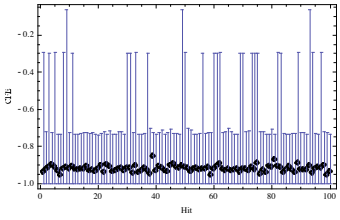
Algorithm 2



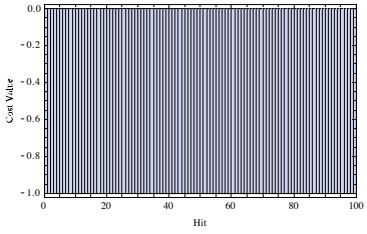


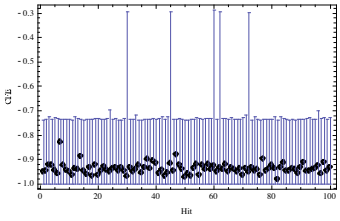
Algorithm 3



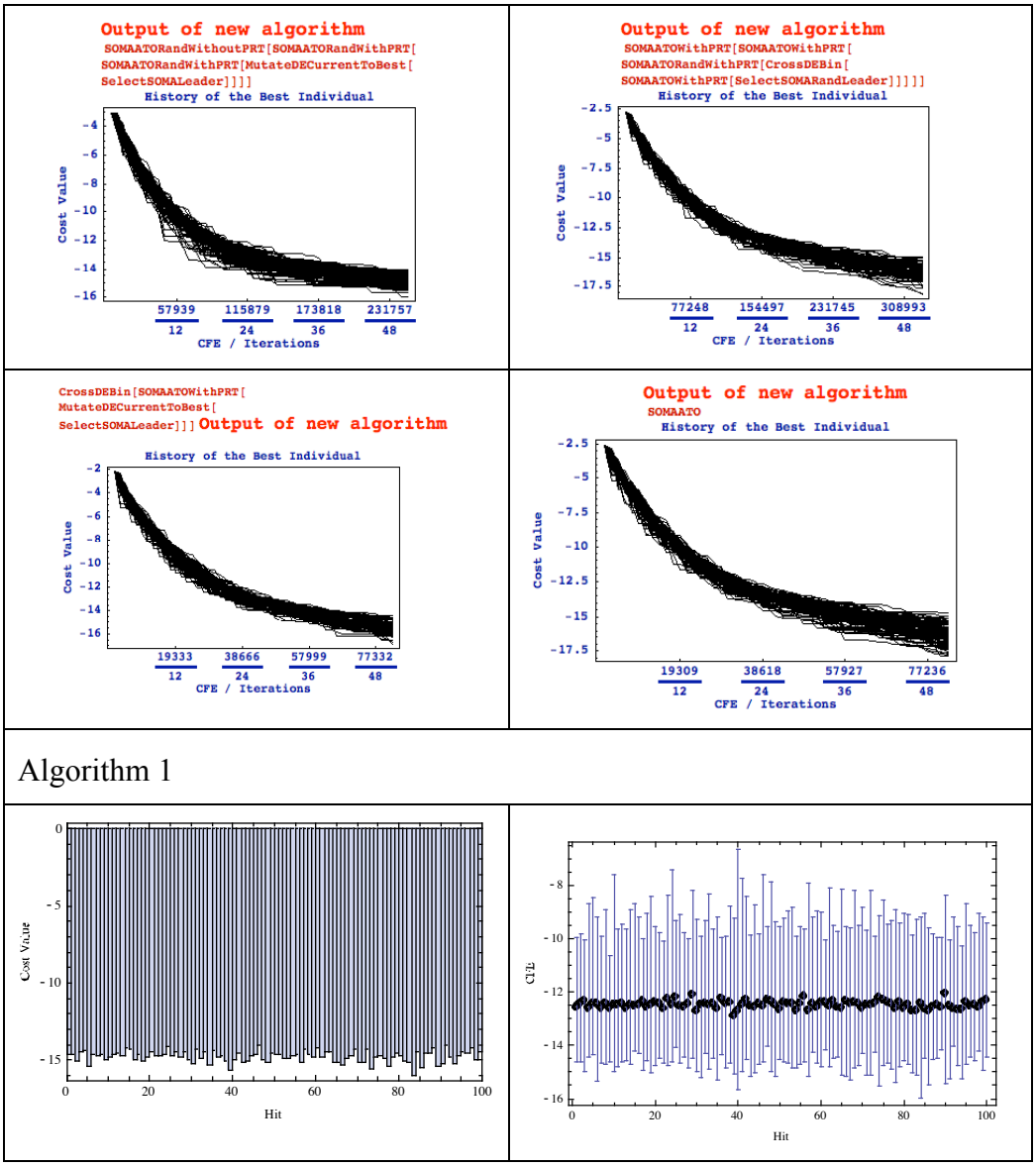


SOMAATO



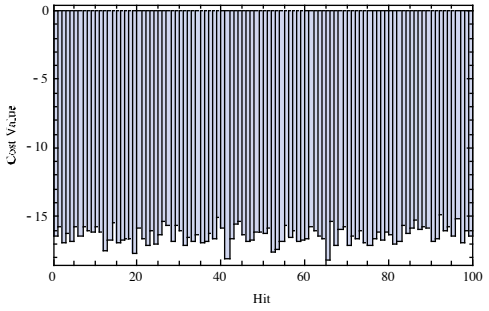


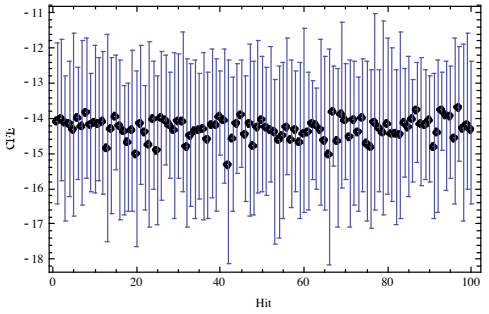
7.47. Master's cosine wave function – 20D



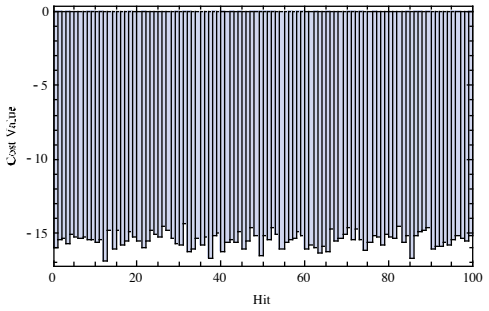
Algorithm 1

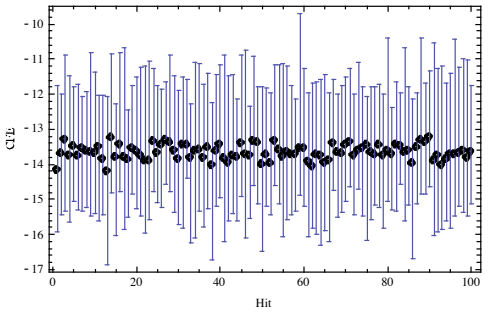
Algorithm 2



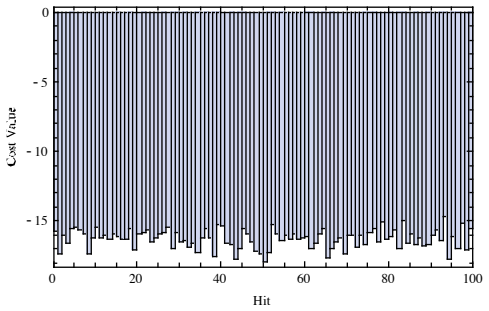


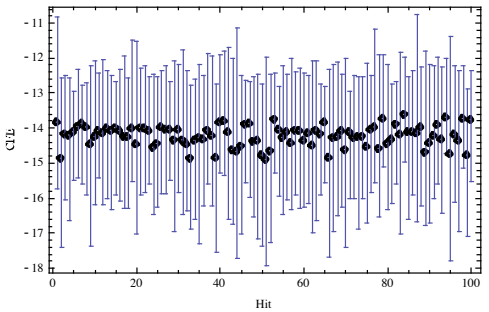
Algorithm 3



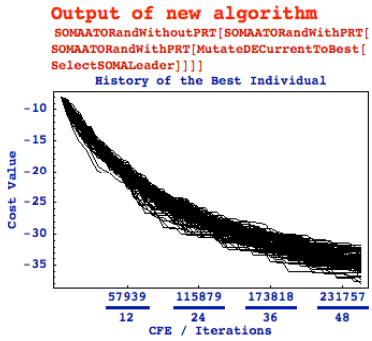


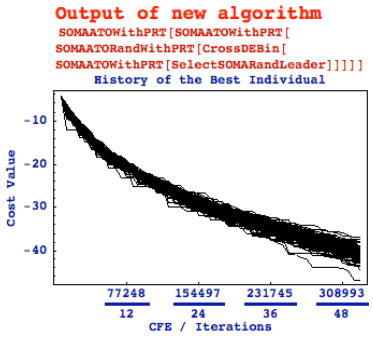
SOMAATO

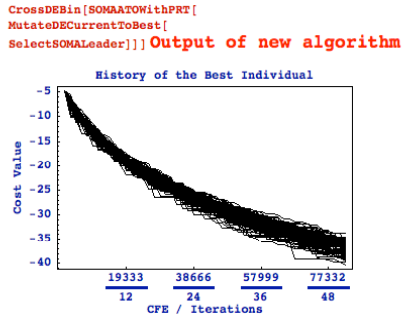


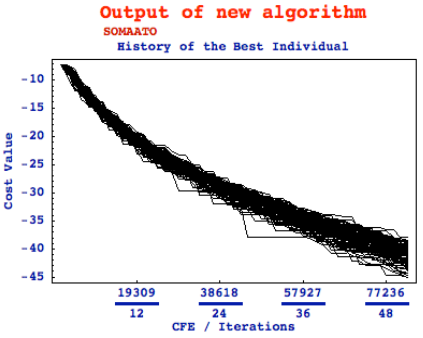


7.48. Master's cosine wave function – 100D

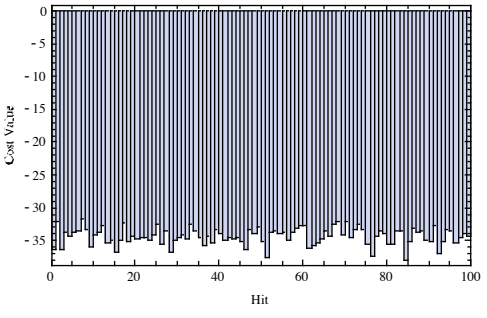


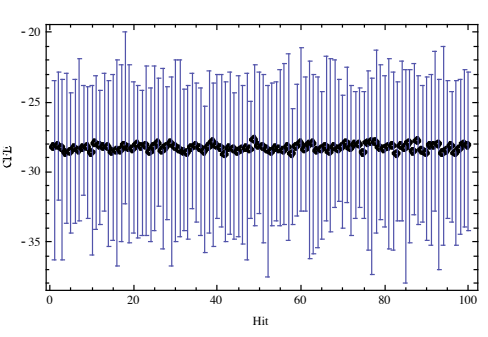




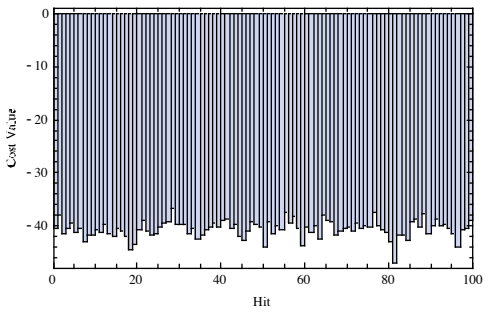


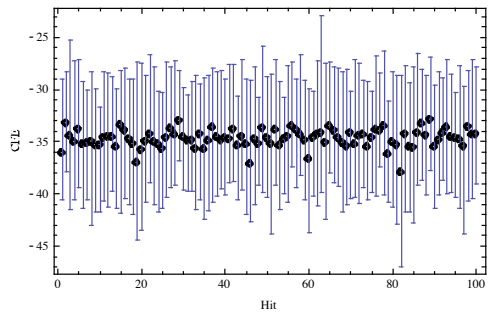
Algorithm 1



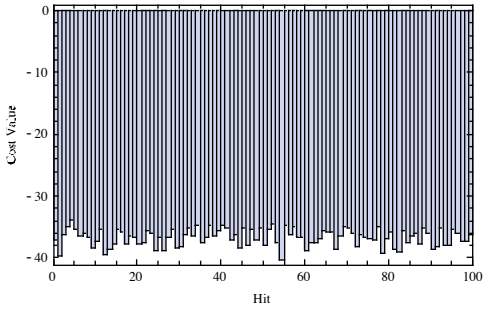


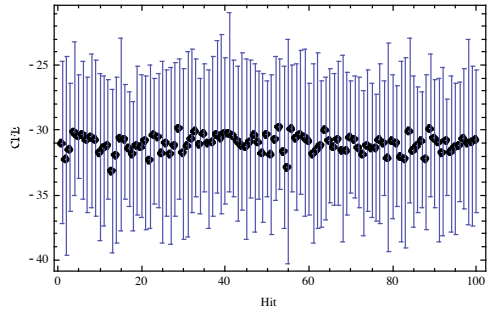
Algorithm 2



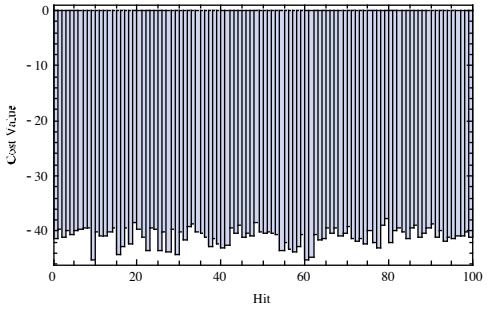


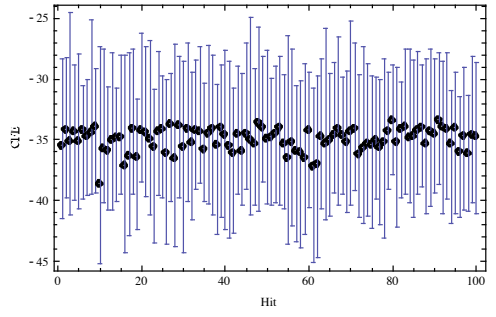
Algorithm 3





SOMAATO





LIST OF AUTHOR'S PUBLICATION ACTIVITES

Journal articles

1) ZELINKA I.,OPLATKOVA Z, NOLLE L..: *Boolean Symmetry Function Synthesis by Means of Arbitrary Evolutionary Algorithms-Comparative Study*, International Journal of Simulation Systems, Science and Technology, Volume 6, Number 9, August 2005, pages 44 - 56, ISSN: 1473-8031, online http://ducati.doc.ntu.ac.uk/uksim/journal/Vol-6/No.9/cover.htm, ISSN: 1473-804x

Conference papers

1) ZELINKA I., OPLATKOVA Z.: *Analytic Programming – Comparative Study*, Mendel '03, In: Proc. 9th International Conference on Soft Computing Mendel'03, Brno, Czech Republic, 2003, 86-89, ISBN 80-214-2135-5
2) ZELINKA I., OPLATKOVA Z.: *Analytic Programming – Comparative Study.* CIRAS'03, The second International Conference on Computational Intelligence, Robotics, and Autonomous Systems, Singapore, 2003, ISSN 0219-6131
3) ZELINKA I.,OPLATKOVA Z.: *Symbolic Regression by Means of Analytic Programming*, Summer School Datastat03 ,Proceedings, Folia Fac.Sci.Nat.Univ.Masaryk.Brunensis, Mathematica 15, Brno,2003, ISBN 80-210-3564-1
4) ZELINKA I.,OPLATKOVA Z.: *Boolean Parity Function Synthesis by Means of Arbitrarry Evolutionary Algorithms-Comparative Study*", In: 8th World Multiconference on Systemics, Cybernetics and Informatics (SCI 2004), Orlando, USA, in July 18-21, 2004, ISBN 980-6560-13-2
5) ZELINKA I.,OPLATKOVA Z.: *Boolean Symmetry Function Synthesis by Means of Arbitrary Evolutionary Algorithms-Comparative Study*, In: Mendel '04, Proc. 10th International Conference on Soft Computing Mendel'04, Brno, Czech Republic, 2004, ISBN 80-214-2676-4
6) ZELINKA I.,OPLATKOVA Z, NOLLE L..: *Boolean Symmetry Function Synthesis by Means of Arbitrary Evolutionary Algorithms-Comparative*

Study, ESM '2004, In: Proc. 18th European Simulation Multiconference, Magdeburg, Germany 2004, ISBN: 3-936150-35-4

7) OPLATKOVA Z.: *Analytic Programming – Boolean Symmetry Problems by means of Evolutionary Algorithms*, In: Proc. PAD2004, Bratislava, Slovensko,2004, ISBN 80-969202-0-0
8) OPLATKOVA Z., ZELINKA I.: *Investigation on Shannon – Kotelnik Theorem Impact on SOMA Algorithm Performance,* ECMS '2005, In: Proc. 19th European Conference on Modelling and Simulation, Riga, Latvia 2005, pages 66-71, ISBN: 1-84233-112-4
9) OPLATKOVA Z., *Optimal Trajectory of Robots Using Symbolic Regression*, In: CD-ROM of Proc. 56th International Astronautical Congress 2005, Fukuoka, Japan, 2005, paper nr. IAC-05-C1.4.07
10) OPLATKOVA Z., ZELINKA I., *Santa Fe Trail for Artificial Ant with Simulating Annealing – preliminary study*, ECMS 2006, In Proc. 20th European Conference on Modelling and Simulation 2006, Bonn, Germany, 28-31 May 2006, pages 56-61, ISBN 0-9553018-0-7
11) ZELINKA I., VOJTESEK J., OPLATKOVA Z.: *Simulation Study of the CSTR Reactor for Control Purposes*, ECMS 2006, In Proc. 20th European Conference on Modelling and Simulation 2006, Bonn, Germany, 28-31 May 2006, pages 479-485, ISBN 0-9553018-0-7
12) OPLATKOVA Z., ZELINKA I., *Creating Evolutionary Algorithms by means of Analytic Programming – Preliminary Study*, Mendel 2006 – 12th International Conference on Softcomputing, Brno, Czech Republic, 31 May – 2 June 2006, pages 19 – 24, ISBN 80-214-3195-4
13) ZELINKA I., VARACHA P., OPLATKOVA Z., *Evolutionary Synthesis of Neural Network*, Mendel 2006 – 12th International Conference on Softcomputing, Brno, Czech Republic, 31 May – 2 June 2006, pages 25 – 31, ISBN 80-214-3195-4
14) OPLATKOVA Z., ZELINKA I., *Investigation on Artificial Ant using Analytic Programming*, GECCO 2006, Seattle, Washington, USA, 8 – 12 July 2006, ISBN 1-59593-186-4
15) OPLATKOVA Z., ZELINKA I., *Setting an Optimal Trajectory By Means Of Analytic Programming*, TMT 2006, Lloret de Mar, Spain, 10 – 15 September 2006, ISBN 9958-617-30-7
16) OPLATKOVA Z., ZELINKA I., *Learning of robots via symbolic regression and evolutionary computation*, workshop WETDAP 2007 in Znalosti 2007, VSB Ostrava, Czech Republic, 2007, pages 27 – 34, ISBN 978-80-248-1332-5

17) OPLATKOVA Z., ZELINKA I., *Santa Fe Trail for Artificial Ant with Analytic Programming and Three Evolutionary Algorithms,* AMS 2007, Phuket Thailand, 27-30 March 2007, pages 334-339, ISBN: 0-7695-2845-7

18) OPLATKOVA Z., ZELINKA I., *Creating evolutionary algorithms by means of analytic programming – design of new cost function*, ECMS 2007, Praha, 3 – 6 June 2007, pages 271 – 276, ISBN 978-0-9553018-2-7, **best paper award**

19) OPLATKOVA Z., ZELINKA I., *Symbolic regression and evolutionary computation in setting an optimal trajectory for a robot*, workshop ETID 2007 in DEXA 2007, 3-7 Sept. 2007, Regensburg, Germany, pages 168-172, ISBN: 978-0-7695-2932-5

CURRICULUM VITAE

PERSONAL INFORMATION		
	Name	Zuzana Oplatková
	Date of birth	22 May 1980
	Present address	Světlov 65, 76302 Zlín - Malenovice
	Marital status	Single
	Contact	tel: +420 604 741 178, email: oplatkova@fai.utb.cz
EDUCATION		
	1994 – 1998	High School in Otrokovice GCE in Czech language, German language, English language, Mathematics and Programming
	1998 – 2003	Tomas Bata University in Zlín at Chemical and Process Engineering program, in the field of Automation and Control Technology in Consumer Industry
	June 2003	State exam and defence of diploma thesis Analytic Programming with excellent mark
	September 2003 - November 2004	Student of a doctoral program in Technical Cybernetics in full-time study
	Since December 2004	Further study in the doctoral program as a part-time student
SCHOLARSHIPS AND ATTENDENCE AT CONFERENCES		
	October – December 2002	Scholarship under the program ERASMUS at The Open University, Oxford Research Unit, Oxford, Great Britain.
	April - July 2004	Scholarship under the program Nonlinear and adaptive control, Politecnico di Milano, Milano, Italy
	2003 – 2007	Attendance at international conference: Mendel'03 and Mendel'06 in Brno, Czech Republic; PAD in Bratislava, Slovakia; ESM in Riga, Latvia; IAC 2005 in Fukuoka, Japan under sponsorship of ESA; ECMS 2006 in Bonn, Germany; GECCO 2006 in Seattle, Washington, USA; TMT 2006 in Lloret de Mar, Spain; AMS 2007 in Phuket, Thailand; EMCS 2007 in Prague, Czech Republic, ETID 2007 (DEXA 2007) in Regensburg, Germany

AWARDS		
	2003	Award for the best diploma thesis
	2007	Award for the best paper at conference ECMS 2007
INVITED LECTURES		
	February 2006	Planetárium Praha, Čeští studenti v ESA: Astronautický kongres Fukuoka 2005 (S mravencem do Fukuoky)
	April 2006	Lecture in Lappeenranta, Finland under ERASMUS / SOCRATES programme, Optimal setting trajectory by means of Analytic Programming
	December 2006	3 lectures in Bielsko – Biala and Cracow, Poland
MEMBERSHIP		
	reviewer of SCI 2005, Florida, USA – 6 articles	
	programme chair of student section and IPC member at ECMS 2006 and ECMS 2007	
	general programme chair for ECMS 2007, Prague, Czech Republic programme co-chair for ECMS 2008, Nicosia, Cyprus	
	biographical profile included in 2007 Edition of Who's Who in the World	
LANGUAGE KNOWLEDGE		
	Czech language	Native
	German language	GCE at High school
	English language	GCE at High school, FCE – December 2005
	French language	Basic
	Italian language	Basic
	Chinese language	Beginner
EMPLOYMENT		
	Since Dec 1, 2004	Lecturer at Tomas Bata University in Zlín - seminars and laboratories – applied informatics, artificial intelligence (neural networks, evolutionary algorithms, fuzzy logic), electronic preparation of documents and www pages
OTHER ACTIVITIES		
	2000 – 2003	I worked as a part-time teacher at the House of Children and Youth "Sluníčko" in Otrokovice. I led 3 to 6 groups of children aimed to basics of work with PC at 8th Basic School in Malenovice. In 2002 I led a course for adults.
	2001 – 2005	Computer administrator at 8th Basic School in Zlín – Malenovice as a part-time job