Improving Differential Evolution using Inductive Programming

Master's Thesis in Applied Computer Science

Marius Geitle

May 15, 2017
Halden, Norway

Abstract

Differential Evolution (DE) has emerged as one of the most powerful and versatile global numerical optimizers for non-differentiable, multimodal problems. The popularity of DE has led to extensive work on improving the algorithm, and significant advances are increasingly more difficult to achieve.

Most researchers seek to improve DE by studying the algorithm using formal analysis, or manual exploration to find improvements. However, many improvements might not be found using these approaches. In this thesis, we use another evolutionary algorithm, the inductive programming system Automatic Design of Algorithms Through Evolution (ADATE), to empirically search for these improvements by systematically testing millions of synthesized modifications.

However, even the rigorous requirements for competitions among state-of-the-art algorithms leave considerable statistical uncertainty on many problems. This presents a significant challenge in how modifications can be evaluated quickly enough. Additionally, the algorithms are too large to improve all-at-once, thereby raising the question of which parts to improve. These questions were explored in three experiments.

The first experiment attempts to improve the mutation operator in the canonical DE algorithm, by quickly evaluating modifications, at the cost of high statistical uncertainty. While the resulting mutation operator performed worse, when tested using a larger number of generations, it provided valuable knowledge for the next experiments.

The next experiment significantly enhanced the statistical certainty when trying to improve the pool of strategies in Competitive Differential Evolution, a variant that uses a pool of competing strategies to produce mutated individuals. The improved pool achieved a significant performance increase when tested on the CEC 2014 benchmark problems.

This led to the third experiment, which resulted in the improvement of the state-of-the-art LSHADE-EpSin algorithm. This DE variant uses an ensemble of sinusoidal functions to generate the scaling parameter F in the current-to-pbest mutation operator. ADATE improved both the mutation operator and the ensemble simultaneously, and found an improved ensemble consisting of a single sine wave that achieves statistically better performance on CEC 2014 problems with 30 dimensions.

Finally, this thesis proposes a specialized version of DE to train Spiking Neural Networks (SNNs). These networks have numerous advantages over traditional artificial neural networks, but no good training methods currently in exists. While no experiment was conducted, the proposed algorithm outlines several modifications that utilize SNNs-specific methods, in a manner that might make the search handle the high number of parameters.

Keywords: Differential Evolution, ADATE, Inductive Programming, SHADE, SHADE-EPSin.

Acknowledgments

I would like to thank my supervisor Jan Roland Olsson. I appreciate all the contributions he has made to make my Master thesis experience stimulating and productive. His enthusiasm was contagious and helped motivate me even through poor results. He helped me see that failed experiments can sometimes provide more knowledge than successful ones, and that every failure is an opportunity to learn and improve.

I would also like to thank my mother and father who has supported me in all my pursuits.

Marius Geitle, 15.05.2017

Contents

List of Figures

List of Tables

Listings

List of Abbreviations

ADATE Automatic Design of Algorithms Through Evolution.

ANN Artificial Neural Networks.

BCM Bienenstock-Cooper-Munro.

BPSL Biologically Plausible Supervised Learning.

CDE Competitive Differential Evolution.

CEC IEEE Congress on Evolutionary Computation.

DE Differential Evolution.

EC Evolutionary Computation.

GA Genetic Algorithms.

GP Genetic Programming.

I&F Leaky Integrate and Fire.

IP Inductive Programming.

LSM Liquid State Machines.

LSTM Long Short-Term Memory.

LTD Long-Term Depression.

LTP Long-Term Potentiation.

ODE Ordinary Differential Equation.

PSO Particle Swarm Optimization.

RNN Recurrent Neural Network.

SNN Spiking Neural Network.

SRM Spike Response Model.

STDP Spike-Timing-Dependent-Plasticity.

Chapter 1

Introduction

1.1 Motivation

Problems requiring finding the optimal parameters of some non-differentiable processes arise in many disciplines such as scientific, engineering, and financial computations. When these problems also become high dimensional, non-convex, and multimodal, solving them becomes a significant challenge. The need for efficient methods has given rise to extensive literature covering both theoretical analysis and practical methods for solving these problems.

The importance of non-differentiable optimization has led to many different methods being developed over the years such as Tabu search[1], Simulated annealing[2], and various Evolutionary Computation (EC) algorithms like the Artificial bee colony algorithm[3], Particle swarm optimization[4], Genetic algorithms[5] and Evolution strategies[6]. Among these, Differential Evolution (DE), introduced by Storn and Price [7] in 1995, has shown itself to be one of the most powerful and versatile numerical optimizers. DE has been successfully used on a large number of practical problems spanning many fields, including space trajectory optimization[8], ocean glider path planning [9], and optimizing fossil fuel burning power plants [10]. In addition to this, variants have been among the best-performing algorithms in most CEC competitions[1]. The popularity of the algorithm makes improving it a very competitive endeavor with many new variations published every year and significant improvements are becoming increasingly harder to achieve.

This thesis investigate whether a related field, Inductive Programming (IP), can introduce new methods for improving DE and its variants. IP is a field dealing with the problem of automatically designing algorithms to solve problems using only an incomplete description. A state-of-the-art IP system is Automatic Design of Algorithms Through Evolution (ADATE)[11]–[13]. ADATE has been shown to be excellent at improving heuristics for tasks such as solving Boolean satisfiability problems, classifying edges in images and improving graph based image segmentation [14]–[16].

Spiking Neural Networks (SNNs) are the third generation of neural networks. Unlike in traditional Artificial Neural Networks (ANN), the neurons in an SNN communicate using binary events called spikes, which are produced when the membrane potential of a neuron reaches a threshold. These networks have several benefits such as being much more power efficient when implemented in hardware[17], and have been shown to have the same computational power using fewer gates [18]. While theoretically interesting, is

[1]For details, see: `http://www.ntu.edu.sg/home/epnsugan/index_files/cec-benchmarking.htm`

has few applications in engineering contexts, apart from some limited use of Liquid State Machines (LSM)[19]. The lack of applications is due to there currently not existing any good training methods, but DE algorithms could likely be used to train SNNs, especially if specialized for these types of networks.

1.2 Research questions

This thesis aims at contributing to the understanding of how IP systems could be used to automatically improve the performance of DE based optimization algorithms. Moreover, we would also like to contribute to the understanding of how specialized versions of DE could be created for training of SNNs.

This leads us to defining the following research questions:

RQ1: How can improvements be evaluated quickly enough for ADATE to test millions of programs while maintaining a sufficiently high statistical certainty?

RQ2: Which parts of the algorithms should be improved? Many of the algorithms have multiple parts, all of which could yield significant improvements, but the algorithms are too large for ADATE to improve all-at-once.

RQ3: How can versions of DE, specialized for training SNNs be created?

RQ1 is especially challenging as there is even high significant statistical uncertainty on many problems when following the rigorous submission requirements for competitions among state-of-the-art optimization algorithms.

1.3 Method

For the experiments aiming to improve variations of DE, the method consists of the following steps:

- Implement the DE variant in Standard ML, and validate that the algorithm is correctly implemented by comparing against either published results or published implementations if available.
- Identify which part or parts of the algorithm the ADATE system should try to improve and represent the problem for ADATE.
- Validate the improved algorithm found by ADATE using additional testing problems.

For understanding how IP systems might be used to create specialized versions of DE for SNNs, an in-depth literature review was used in combination with knowledge from experiments to propose an algorithm that could be used to train SNNs.

1.4 Structure of this thesis

This section briefly summarizes the contents of each of the remaining chapters in this thesis. The first three chapters are devoted to discussing background literature on optimization, the DE algorithm, and the field of IP. The next three discuss three experiments for improving three variations of DE using ADATE. The next then discusses SNNs, and how to create specialized versions of DE. The second to last chapter proposes some future work, and the last chapter concludes this thesis.

Chapter 2 formulates the optimization problem and discusses both derivative and derivative-free optimization. The chapter also reviews some of the most important methods for non-differentiable optimization. Lastly, the chapter covers how optimization algorithms might be evaluated by discussing problem sets and the Wilcoxon rank-sum test.

Chapter 3 provides an extensive description of the DE algorithm and reviews some of the work that has been done on using the algorithm to solve practical problems and understand how the basic steps of mutation, crossover, and selection affect the search. It also reviews some techniques for parameter-control and Memetic variations of DE.

Chapter 4 introduces the concept of inductive programming and discusses the differences between analytical and search based IP systems. It also provides an in-depth description of ADATE.

Chapter 5 discusses an experiment where we tried to improve the mutation operator DE/rand/1 in the original DE algorithm. The evaluation followed a quantity over quality approach with the focus on evaluating improvements quickly at the cost of low statistical certainty. While the improved algorithm ended up adapting to the low number of generations used during testing candidate programs, the experiment provided valuable knowledge for future experiments.

Chapter 6 reports on an experiment that significantly enhanced the statistical certainty of the evaluation used in ADATE at the cost of requiring much more computation time to evaluate each improvement. This resulted in the successful improvement of the pool of mutation strategies in b3e3pbest, a version of DE developed by Bujok, Tvrdk, and Polakova [20]. The variant adapts to the function being optimized by probabilistically selecting a mutation strategy such that the most successful are used most often. However, an analysis of the performance of the programs reveal significant problems in the mechanism used to select the most successful strategy, which likely prevents the algorithm from achieving better results than the best DE variants.

Chapter 7 reports on the third experiment in which we further improved the statistical certainty of the performance evaluation in ADATE by increasing the number of problems used for training and increasing the repetitions of each problem. This led to the successful improvement of the mutation heuristics in the state-of-the-art DE variant LSHADE-EpSin developed by Awad, Ali, Suganthan, *et al.* [21]. The improved heuristics achieved statistically significant improved performance on CEC 2014 problems of 30 dimensions with a much smaller algorithm.

Chapter 8 provides an in-depth discussion on SNN and discuss how to create versions of DE specialized for training SNNs.

Chapter 9 proposes further work based on all three experiments, and discuss other methods for creating specialized versions of DE.

Chapter 10 concludes this thesis, summarizes the main contributions of the work and outline possible directions for future research.

The thesis ends with an appendix containing the definitions for the synthetic benchmark functions used as part of training and validation sets.

Chapter 2

Optimization

Optimization is the process of finding optimal parameters with respect to some objective or objectives. In this thesis, only the problems consisting of real parameters with a single real objective function are considered. That is, problems that can be stated as follows:

Given an objective function:

$$f : B \subseteq \mathbb{R}^D \longrightarrow \mathbb{R} \tag{2.1}$$

with B being a D-dimensional hyperbox formed by the bounds:

$$\mathbf{x_{min}} = (x_{min,1}, x_{min,2}, \cdots, x_{min,D}) \tag{2.2}$$

$$\mathbf{x_{max}} = (x_{min,1}, x_{max,2}, \cdots, x_{max,D}), \tag{2.3}$$

find an element $\mathbf{x_0}$ such that $f(\mathbf{x_0}) \leq f(\mathbf{x})$ for all $\mathbf{x} \in B$.

This definition limits the optimization problems to minimization. This is intentional as the conversion between minimization and maximization is, in most cases, trivial. Most techniques used to solve optimization problems like this can be classified as being either derivative based or derivative free.

2.1 Derivative-based optimization

Derivative-based optimization uses information about the gradient during optimization to quickly converge to a locally optimal solution. While it is possible to use the gradient directly for some simple functions to find the true minimum by solving for $f'(x) = 0$, most problems require using the gradient iteratively to find the minimum.

The simplest of these is the gradient descent method:

$$\mathbf{x}^{(t+1)} = \mathbf{x}^{(t)} - \gamma \cdot f'(\mathbf{x}^{(t)}) \tag{2.4}$$

with γ being a parameter controlling the step size and the gradient is defined as

$$f'(x) = \nabla f(\mathbf{x}) = \begin{bmatrix} \frac{\partial f(\mathbf{x})}{\partial x_1} \\ \frac{\partial f(\mathbf{x})}{\partial x_2} \\ \vdots \\ \frac{\partial f(\mathbf{x})}{\partial x_D} \end{bmatrix}. \tag{2.5}$$

The importance of the gradient descent method has led to the development of many variations that solve or mitigate some of the practical problems with steepest descent. But all these variations require the objective function to be differentiable.

2.2 Derivative-free optimization

It is in many situations not possible to differentiate the objective function. For example, if the objective function is a computer program with branching, calculating the symbolic derivative might be impossible or prohibitively difficult.

Without the derivative, the algorithms have to search through the search landscape using only information obtained by generating and testing a, sometimes large, number of points. The simplest of these is brute force search which tests every point in a grid covering the search space. This method is simple to implement. But the computation time quickly becomes prohibitively large as the step sizes in the grid are reduced to achieve higher accuracy. Random walk is another simple method which works by starting with a single point and adding a random value sampled from a Gaussian distribution to each dimension. This improves slightly on brute force search but still suffers from the difficulty of finding the optimal solution, even for unimodal problems.

Many solutions have been proposed for derivative-free optimization over the years. We briefly discuss some of the most important in the following sections.

2.2.1 Simulated annealing

Simulated annealing is an optimization method based on the technique of heating and cooling a metal to increase the size of its crystals[2]. When the metal is hot, it is easier to change the structure of the metal because the crystals are freer to move around. As the metal cools, the crystals become increasingly constricted in their movement.

Drawing inspiration from this process, Simulated annealing maintains two pieces of information: the temperature and the state, usually a candidate solution. The algorithm then iteratively perturbs and replaces the candidate solution with better solutions as they are discovered. The size of the perturbation is determined by the temperature which starts high and is gradually lowered for each iteration.

2.2.2 Tabu search

Tabu search is a single individual, local search algorithm developed by Glover [1]. The algorithm moves the individual to the best solution in the neighborhood. Moves to worse solutions are allowed, stopping only when a criterion is satisfied.

The search as described above might result in infinite loops; these are prevented by storing previous solutions for some time in a data structure called a Tabu list. By not allowing moves to any candidate in the list, the algorithm is free to explore the search space in a greedy manner.

2.2.3 Evolutionary computation

Inspired by the biological process of evolution, evolutionary computation algorithms start with a population of candidate solutions which are iteratively evolved towards better solutions by using some or all of the fundamental mechanisms, namely, mutation, crossover, and selection.

Artificial bee colony algorithm

The Artificial bee colony algorithm is an optimization algorithm inspired by the foraging behavior of bees as proposed by Karaboga [3]. The algorithm starts with a population of food sources, represented by randomly initialized candidate solutions.

The algorithm consists of three phases, each of which governs the process of the three types of bees, namely, employed bees, onlooker bees and scout bees.

Employed bees are assigned a food source and explore the neighborhood of that food source in search of better food sources.

Onlooker bees then get probabilistically assigned food sources weighted according to how good each food source is after the employed bees finish.

Scout bees explore the search space randomly in search of new and better food sources.

The best food sources are then saved and the process repeated until a termination criterion is satisfied.

Particle swarm optimization

Particle Swarm Optimization (PSO), first proposed by Eberhart and Kennedy [4] is an optimization algorithm inspired by how animals like birds, and fish are capable of solving complicated problems without central control.

The algorithm uses a set of candidate solutions called particles which together are known as a swarm. At each iteration t, the position of the particle i are updated according to:

$$\mathbf{x_i^{(t+1)}} = \mathbf{v_i^{(t+1)}} + \mathbf{x_i^{(t+1)}} \tag{2.6}$$

The velocity $\mathbf{v_i^{(t+1)}}$ is calculated according to:

$$\mathbf{v_i^{(t+1)}} = \mathbf{v_i^{(t)}} + c_1(\mathbf{p_i^{(t)}} - \mathbf{x_i^{(t)}})\mathbf{R_1^{(t)}} + c_2(\mathbf{g^{(t)}} - \mathbf{x_i^{(t)}})\mathbf{R_2^{(t)}} \tag{2.7}$$

Here momentum is maintained by basing the new value on the previous velocity $\mathbf{v_i^{(t)}}$. The parameter c_1 controls how much the trajectory of the particle is affected by the best point $\mathbf{p_i^{(t)}}$, found so far by the particle, and c_2 controls how much the trajectory is affected by $\mathbf{g^{(t)}}$, the best point found by any particle. R_1 and R_2 are two diagonal matrices with random numbers drawn from a uniform distribution between 0 and 1.

By repeating the application of Equation (2.6) and Equation (2.7), incrementing t and updating the best known points at each iteration, the population of particles will converge to a local optimum over time.

The standard algorithm here has multiple problems. The amounts of identified problems and solutions are vast, and a complete review would be prohibitively large, but a recent review of the most practically relevant issues and their solutions can be found in [22].

Genetic algorithms

Genetic Algorithms (GA), proposed by Holland [5], solve optimization problems by a process inspired by natural selection. The basic outline of the algorithm is as follows:

Algorithm 1 General structure of a genetic algorithm

```
Initialize population with random candidate solutions
while Termination criteria not met do
    Select parents.
    Recombine pair of parents.
    Mutate the resulting offspring.
    Evaluate new candidates.
    Select next generation.
end while
```

The individuals are encoded using binary encoding. That is, each individual is represented by a string of 0's and 1's. This representation is practical for problems which lends themselves to a binary representation, but makes solving problems with real values difficult.

Evolution strategies

Evolution strategies is an evolutionary algorithm similar to genetic algorithms but designed to solve real-valued problems[6]. Here the individuals are represented using two pieces of information. The real-valued vector representing the candidate solution, and a vector of parameters which determine how the individual is mutated. The parameters are evolved alongside the solution as a form of adaptation.

2.3 Evaluating optimization algorithms

Most publications of optimization algorithms use a self-defined set of synthetic functions with their own constraints on the number of function evaluations, search domain range and initialization. This practice often makes it impossible to compare different algorithms using the published results.

A solution comes from IEEE Congress on Evolutionary Computation (CEC) which has designed several predefined sets of benchmark problems with strict rules for how results are to be reported to make the results directly comparable as part of competitions. For single-objective, real-valued optimization, CEC covered four different classes as part of the 2016 competition[1]:

Single parameter-operator set based case Optimization of problems with a single objective and real-valued parameters. The optimizer is allowed to perform a large number of function evaluations using a single configuration across all problems. This set is referred to as CEC 2014 in the remainder of this thesis [23].

Learning-based case Similar to CEC 2014, but the researcher is allowed to adjust the parameter for each problem to find the optimal solution.

[1]Detailed descriptions can be found at: `http://www.ntu.edu.sg/home/EPNSugan/index_files/CEC2016/CEC2016.htm`

Multi-solution niching case The algorithm has to find as many as possible of the optima of a given fitness landscape, not just a single global optimum.

Computationally expensive case Similar to CEC 2014, but with a much smaller budget of allowed function evaluations.

2.3.1 CEC 2014 problems

The CEC 2014 set consists of 30 unique minimization problems all defined for $D = 10$, $D = 30$, $D = 50$ and $D = 100$ dimensions. In this project, the CEC 2014 problems with 10 and 30 dimensions were used to evaluate all improved algorithms. When computation time permitted, problems with 50 dimensions were also used.

The requirements include reporting the best, worst, median, mean and standard deviation for all functions. The optimizers must start from a population randomly initialized from a uniform distribution using a time-dependent seed and repeated 51 times. The optimizer is allowed to evaluate the fitness function a maximum of $10000 \cdot D$ times[23], the search space is limited to $[-100, 100]^D$. The problem is considered solved when an error value smaller than 10^{-8} is achieved. At which point the search can be terminated. All values smaller than 10^{-8} should be treated as zero.

To make the problems unique and slightly more challenging, CEC 2014 has used three randomization strategies:

shift All functions are shifted by a shift provided in precomputed data files for each function. The shift has been computed by randomly sampling from $[-80, 80]$.

rotation Some functions are rotated using a precomputed rotation matrix. The exact algorithm used to compute the rotation matrices is poorly documented.

permutation The components of all functions have been randomly permuted.

2.3.2 Wilcoxon rank-sum test

A statistical test is needed to preform statistically sound comparisons between optimization algorithms, but the results are rarely normally distributed. Thereby making parametric methods such as the two-sample t-test unsuitable. Instead, the Wilcoxon rank-sum test is therefore used, as it does not have the assumption of normality on the observations[24]. This test uses the ordering of the observations within two samples to determine whether there are statistically significant differences between the two samples.

Chapter 3

Differential evolution

DE is similar to classical evolutionary algorithms in which a population of candidate solutions, initialized to a uniform sampling of the instance space, are gradually improved by repeatedly modifying the candidates and selecting the next generation. However, unlike other evolutionary algorithms such as Evolution Strategies, mutation is accomplished by using the scaled difference between members of the population[6]. This has the effect of adapting the step size to the fitness landscape over time.

DE consists of four phases: initialization, mutation, crossover, and selection. The last three of these are repeated until a termination condition such as the maximum number of generations is met.

Algorithm 2 The standard differential evolution algorithm.

```
$P^{(1)} \leftarrow \{\mathbf{x_i^{(1)}}, \mathbf{x_i^{(1)}}, \dots, \mathbf{x_{N_p}^{(1)}}\}$        ▷ see section 3.1 for details.
$t \leftarrow 1$
while Termination criteria not met do
    for $i \leftarrow 1, N_p$ do
        $\mathbf{v_i^{(t)}} \leftarrow mutate(\mathbf{x_i^{(t)}})$        ▷ see section 3.2 for details.
        $\mathbf{u_i^{(t)}} \leftarrow crossover(\mathbf{x_i^{(t)}}, \mathbf{v_i^{(t)}})$        ▷ see section 3.3 for details.
        if $f(\mathbf{u_i^{(t)}}) \leq f(\mathbf{x_i^{(t)}})$ then
            Insert $\mathbf{u_i^{(t)}}$ into $P^{(t+1)}$
        else
            Insert $\mathbf{x_i^{(t)}}$ into $P^{(t+1)}$
        end if
    end for
    $t \leftarrow t + 1$
end while
```

3.1 Initialization

DE starts by initializing a population of N_p, D-dimensional real-valued vectors. Individuals in the population are denoted by

$$\mathbf{x_i^{(t)}} = \left(x_{i,1}^{(t)}, x_{i,1}^{(t)}, \cdots, x_{i,d}^{(t)}\right), \tag{3.1}$$

with $t = 1, 2, \cdots, t_{max}$ being the generation number.

The initial population should be initialized to cover as much as possible of the search space constrained by the minimum and maximum bounds $\mathbf{x}_{min} = (x_{min,1}, x_{min,2}, \cdots, x_{min,D})$ and $\mathbf{x}_{max} = (x_{max,1}, x_{max,2}, \cdots, x_{max,D})$. The j'th component may then be initialized as:

$$x_{i,j}^{(1)} = x_{min,j} + rand_{i,j}(0,1)(x_{max,j} - x_{min,j}) \tag{3.2}$$

with $rand_{i,j}(0,1)$ being a uniformly random value between 0 and 1.

3.2 Mutation

The next step is to generate a donor vector $\mathbf{v_i^{(t)}}$ for each member of the population using mutation. In all operators, R_1, R_2, R_3, R_4, and R_5 are mutually exclusive, random integers, drawn from the range $[1, N_p]$, and all being different from the base index i. $\mathbf{x_{best}^{(t)}}$ is the individual with the best fitness value for that generation. F is a scaling factor typically in the range $[0.1, 2.0]$.

3.2.1 DE/rand/1

Proposed in the original 1996 paper by Storn [25] and perhaps the most used mutation operator. The DE/rand/1 operator creates a donor vector by adding the scaled difference between two random individuals to a third randomly selected individual:

$$\mathbf{v_i^{(t)}} = \mathbf{x_{R_1}^{(t)}} + F\left(\mathbf{x_{R_2}^{(t)}} - \mathbf{x_{R_3}^{(t)}}\right) \tag{3.3}$$

3.2.2 DE/best/1

The DE/best/1[25], also proposed in the original publication adds the scaled difference between two random individuals to the best individual in the population:

$$\mathbf{v_i^{(t)}} = \mathbf{x_{best}^{(t)}} + F\left(\mathbf{x_{R_1}^{(t)}} - \mathbf{x_{R_2}^{(t)}}\right) \tag{3.4}$$

3.2.3 DE/current-to-best/1

This mutation, also named DE/target-to-best/1 [26] and part of the original publication [25], is usually used with the only crossover being the arithmetic operation on the current individual $\mathbf{x_i^{(t)}}$:

$$\mathbf{v_i^{(t)}} = \mathbf{x_i^{(t)}} + F\left(\mathbf{x_{best}^{(t)}} - \mathbf{x_i^{(t)}}\right) + F\left(\mathbf{x_{R_1}^{(t)}} - \mathbf{x_{R_2}^{(t)}}\right) \tag{3.5}$$

This mutation operator is known to easily get stuck in local optima, especially on high dimensional problems[27]. However, a more robust, less greedy variant is used as part of the SHADE-EPsin algorithm discussed in chapter 7. That version is commonly used in combination with the binomial crossover in most state of the art DE variants. However, that version is not included in this list as, it requires larger modifications to the DE algorithm to include an external archive.

3.2.4 DE/rand/2

DE/rand/2 stabilizes the search by including the difference between another pair of individuals[25]:

$$\mathbf{v_i^{(t)}} = \mathbf{x_{R_1^i}^{(t)}} + F\left(\mathbf{x_{R_2}^{(t)}} - \mathbf{x_{R_3}^{(t)}}\right) + F\left(\mathbf{x_{R_4}^{(t)}} - \mathbf{x_{R_5}^{(t)}}\right) \tag{3.6}$$

3.2.5 DE/best/2

Similar to DE/rand/2, but uses the best individual as the base[25]:

$$\mathbf{v_i^{(t)}} = \mathbf{x_{best}^{(t)}} + F\left(\mathbf{x_{R_1}^{(t)}} - \mathbf{x_{R_2}^{(t)}}\right) + F\left(\mathbf{x_{R_3}^{(t)}} - \mathbf{x_{R_4}^{(t)}}\right) \tag{3.7}$$

3.2.6 DE/Randrl/1

Designed to speed up convergence when using DE/rand/1, but reduce the problems of getting stuck in local optima which are common with DE/best/1. DE/Randrl/1 use the tournament best as the base individual[28]:

$$\mathbf{v_i^{(t)}} = \mathbf{x_{tb}^{(t)}} + F_i^{(t)}\left(\mathbf{x_{p_2}^{(t)}} - \mathbf{x_{p_3}^{(t)}}\right), \tag{3.8}$$

where $\mathbf{x_{tb}^{(t)}}$ is the tournament best among three randomly chosen individuals, with $\mathbf{x_{p_2}^{(t)}}$ and $\mathbf{x_{p_3}^{(t)}}$ being the remaining two individuals. $F_i^{(t)}$ is chosen randomly from $F_i^{(t)} \in [-1, -0.4] \cup [0.4, 1]$.

3.2.7 Trigonometric mutation

Fan and Lampinen [29] proposed to use triangulation to more quickly find the optimal solution by forming the donor vector according to:

$$\mathbf{v_i^{(t)}} = \begin{cases} \mathbf{y_i^{(t)}} & \text{if } rand_i^{(t)}(0,1) \leq \Gamma \\ \mathbf{x_{R_1}^{(t)}} + F\left(\mathbf{x_{R_2}^{(t)}} - \mathbf{x_{R_3}^{(t)}}\right) & \text{otherwise} \end{cases} \tag{3.9}$$

With

$$\begin{split}\mathbf{y_i^{(t)}} = \frac{\mathbf{x_{R_1}^{(t)}} + \mathbf{x_{R_2}^{(t)}} + \mathbf{x_{R_3}^{(t)}}}{3} + (p_2 - p_1)\left(\mathbf{x_{R_1}^{(t)}} - \mathbf{x_{R_2}^{(t)}}\right) + \\ (p_3 - p_2)\left(\mathbf{x_{R_2}^{(t)}} - \mathbf{x_{R_3}^{(t)}}\right) + (p_1 - p_3)\left(\mathbf{x_{R_3}^{(t)}} - \mathbf{x_{R_1}^{(t)}}\right)\end{split} \tag{3.10}$$

$$p' = \left|f(\mathbf{x_{R_1}^{(t)}})\right| + \left|f(\mathbf{x_{R_2}^{(t)}})\right| + \left|f(\mathbf{x_{R_3}^{(t)}})\right| \tag{3.11}$$

$$p_1 = \left|f(\mathbf{x_{R_1}^{(t)}})\right| / p' \tag{3.12}$$

$$p_2 = \left|f(\mathbf{x_{R_2}^{(t)}})\right| / p' \tag{3.13}$$

$$p_3 = \left|f(\mathbf{x_{R_3}^{(t)}})\right| / p' \tag{3.14}$$

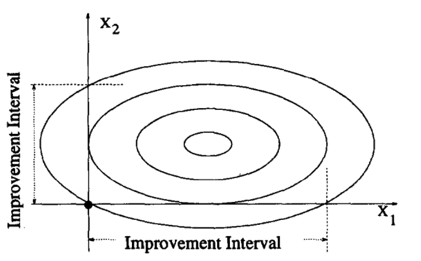
where $f(\mathbf{x})$ is the function to be optimized, and $\Gamma \in [0, 1]$ is the rate which trigonometric mutation should be used, DE/rand/1 is used with a probability of $1 - \Gamma$.

3.3 Crossover

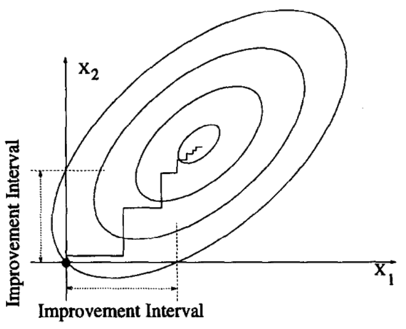
Trial vectors are created by recombining the target vector $\mathbf{x_i^{(t)}}$ and the donor $\mathbf{v_i^{(t)}}$ using a crossover operator. How many components get taken from the donor is controlled by the crossover rate parameter: CR. A good choice of CR is especially important when optimizing non-separable functions[30]. In this thesis, *separable functions* are functions which are *additively separable* such that they be expressed as:

$$f(x_1, x_2, \dots, x_D) = \sum_{j=1}^{D} f_j(x_j) \tag{3.15}$$

Additively Separable functions are special because they can be optimized one variable at a time. As a result, the problems do not get harder as the dimensionality increases.



(a) Improvement interval on a quadratic function.



(b) Improvement interval on a rotated quadratic function.

Figure 3.1: Illustration by Salomon [30] on the importance of crossover in non-separable functions.

The importance of the crossover rate on separable functions was illustrated by Salomon [30] using figure 3.1a. On this non-rotated function, there is a large improvement interval which in turn means there is a high probability of improvement when moving along any of the dimensions in the right direction. However, when looking at the same quadratic function in figure 3.1b, turned non-separable by rotation, the improvement intervals have shrunk. Thus reducing the probability of an improvement when moving along only one of the dimensions. The path in figure 3.1b also illustrates another problem. The number of steps required has increased to a point where the optimum might never be reached.

These observations mean that using a small CR value might work well on separable functions, but will reduce progression of the search on non-separable functions.

Another important work on understanding the crossover rate comes from Zaharie [31], [32] who recognized that the variance before and after mutation and crossover follow the relation:

$$E(Var(U^{(t)}) = c \cdot Var(P^{(t)}), \tag{3.16}$$

with

$$c = 2F^2 p - \frac{2p}{N_p} + \frac{p^2}{N_p} + 1 \tag{3.17}$$

with p being the probability a component is mutated. This observation can be used to determine the behavior of the search. By adjusting CR and F values using c as a guide, the variance of the population can be controlled. Specifically, using $c = 1.07$ will stimulate an 7% increase in the population variance. Kukkonen found empirically that a value in $[1, 1.5]$ performs seemed to perform best [33], [34].

3.3.1 Binomial crossover

The binomial crossover produces a trial vector by selecting a component from the donor vector whenever a randomly generated value drawn from a uniform distribution is below the crossover rate CR. Additionally, a component k is randomly chosen per iteration to always come from a donor vector as follows:

$$u_{i,j}^{(t)} = \begin{cases} v_{i,j}^{(t)} & \text{if } i = k \text{ or } rand_{i,j}(0,1) \leq CR. \\ x_{i,j}^{(t)} & \text{otherwise.} \end{cases} \tag{3.18}$$

An example of this is illustrated in figure 3.2.

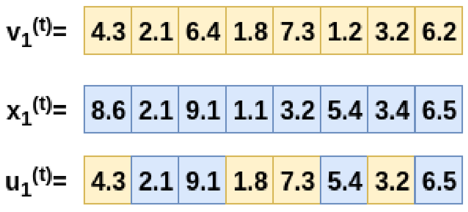


Figure 3.2: Example of the binomial crossover operation.

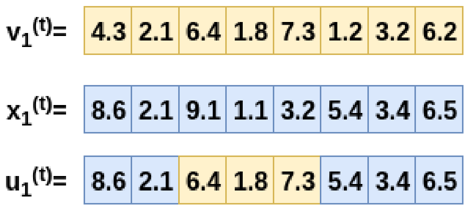


Figure 3.3: Example of the exponential crossover operation.

3.3.2 Exponential crossover

Exponential crossover tries to exploit relationships between adjacent components. It works by choosing a random starting component and selecting the next L consecutive components in a circular manner from the donor vector. The number of components L is calculated as follows:

Algorithm 3 Exponential crossover

```
$L \leftarrow 0$
**repeat**
    $L \leftarrow 0$
**until** $rand(0,1) > CR$ **or** $L > D$
```

An example of this is illustrated in figure 3.3.

Compared to binomial crossover, the significance of the CR value in the exponential crossover is harder to understand. The work of Zaharie [32] eases this by showing that the probability p of a component being chosen from the donor vector follows the relation:

$$CR^D - DpCR + Dp - 1 = 0 \tag{3.19}$$

3.4 Selection

Selecting the survivors to form the next generation is accomplished by pairwise tournament selection:

$$\mathbf{x_i}^{(t+1)} = \begin{cases} \mathbf{u_i} & \text{if } f\left(\mathbf{u_i^{(t)}}\right) \leq f\left(\mathbf{x_i^{(t)}}\right), \\ \mathbf{x_i}^{(t+1)} & \text{otherwise,} \end{cases} \tag{3.20}$$

where $f(\cdot)$ is the objective function to be optimized.

3.5 Parameter control

Parameters have a large impact on the search performance and are known to be problem dependent[35]. They might also need to be changed during the search, either following a fixed predefined plan or adapted during the search.

3.5.1 Population size

In DE, the population size is most often kept constant during the entire optimization. An exception to this is the L-SHADE algorithm proposed by Tanabe and Fukunaga [36]. L-SHADE reduce the population size linearly in terms of the consumed function evaluations. This technique is described in detail in section 7.1.3.

In the field of evolutionary algorithms at large, population size is recognized as one of the most important parameters to control with extensive work on:

- Theoretical studies[37]–[41].
- Adapting the population size to changing circumstances during the run[42], [43].
- Controlling the population size using a predefined plan or searching for optimal population sizes[44], [45].

A comprehensive review is available in [46].

3.5.2 Scaling factor and Crossover rate

In DE, the scaling factor F and crossover rate CR are usually adapted using the same techniques for both parameters, only differing in biases and the sampling distributions.

SaDE, developed by Qin and Suganthan [47], was one of the early attempts at making an adaptive version of DE. The algorithm records successful CR values for five generations; the mean of which is then used for the next five generations to produce CR values by sampling from a normal distribution.

In 2009, Zhang and Sanderson [48] proposed the JADE algorithm which provides improved adaptation mechanics for both CR and F. Instead of maintaining a record of successful values like those used in SaDE, JADE maintains two variables: μ_F and μ_{CR}. These are updated at the end of the generation, using values which were successful in the current generation. μ_F and μ_{CR} can be prevented from changing too quickly by adjusting a parameter.

3.5.3 Adaption of mutation operators

The optimal mutation operator depends on the function being optimized. This can be seen in Tables 5.5 to 5.8, where the performance of the mutation operators differs significantly between the functions. There have been several attempts dynamically switching between different mutation operators during evolution, this section describes some of these.

SaDE switches adaptively between the two strategies “DE/rand/1/bin” and “DE/best/2/bin” by recording how successful each strategy is during a learning period. The strategies are then selected probabilistically by weighting the strategies using the recorded success counts.

A similar algorithm, Competitive Differential Evolution (CDE), developed by Tvrdk [49]. Switches between a pool of strategies, each of which include the scaling factor and crossover rate in addition to the mutation operator. Each strategy is associated with a counter which counts how many times the strategy has been successful. These counts are then used as weights when probabilistically selecting a strategy. The counts reset whenever the probability of selecting any of the strategies becomes too low. The pool of strategies in CDE has been improved in chapter 6.

While both the two preceding algorithms use counts to select a good strategy probabilistically, Mallipeddi, Suganthan, Pan, *et al.* [50] used a different approach. They maintain a dynamic pool of strategies where at the end of each generation, the strategies which have produced individuals that are selected for the new generation are inserted into the pool.

3.6 Memetic variants

Memetic algorithms hybridize evolutionary algorithms by including local improvement heuristics into the default population-based evolutionary algorithms. This section describes some of these approaches applied to DE. Including local search heuristics in DE is relatively common, with many different proposals on how this can be accomplished. In most cases, the local search heuristics is only used after a certain number of generations have passed, or with probabilistic triggering.

Local improvement based

Local improvement based local search tries to improve some or all of the individuals in the population by using some common technique like hill-climbing, or random perturbations.

One such method proposed by Wang, Qian, and Hu [51] uses chaos theory to produce random perturbation of the best 20% individuals after each generation. A similar approach was adopted by Jia, Zheng, and Khan [52], but only applied it on the best individual at each generation.

A proposal by Poikolainen and Neri [53] is to perform a step along each axis on a subset of the individuals in the population, replacing the individual each time an improvement has been found.

Crossover based

Crossover local search based approaches exploit the recombination to produce children in the neighborhood around the individuals in the population.

Ali, Pant, and Nagar [54] propose two crossover based local search methods. The first involves repeatedly applying the trigonometric mutation operator discussed in section 3.2.7 until no further improvement could be found. The second method uses quadratic interpolation to calculate the new individuals. A recent proposal by Peng and Wu [55] uses Taguchi local search to produce new individuals.

Ensemble based

The LSHADE-EpSin algorithm. which is improved in chapter 7, uses a Gaussian random walk on an independent, randomly initialized population to produce new individuals. The best performing individuals found by the local search replace random individuals in the DE population.

3.7 Applications

DE and its variations have been successfully used to solve a vast amount of practical problems. This section lists some of these, for a more extensive review, see [56].

3.7.1 Economic Dispatch problem

DE has been used extensively to ensure optimal power delivery while reducing fuel cost and emission of pollutants in fossil burning power plants while dealing varying power demands. The most common approach is to formulate mathematical models with the goal of incorporating all essential and relevant constraints. A recent, extensive review of this field is available in [10].

3.7.2 Ocean glider path planning

An ocean glider is an autonomous underwater vehicle that propels itself forward by changing its buoyancy. For the glider to reach the target, it must react to changes in ocean currents and the dynamics of the vessel using a power efficient algorithm. Zamuda and Sosa [9] proposed to use DE to plan a three-dimensional path with the fitness being the distance to the target obtained by running a simulation.

3.7.3 Space trajectory optimization

Vasile, Minisci, and Locatelli [8] propose a modified version of DE for multi-objective optimization to solve space trajectory planning problems. They use two main models of space trajectories, one with deep space maneuvers, and one without.

Chapter 4

Inductive programming

IP is the field concerned with automatically designing algorithms from incomplete, often ambiguous specifications of the problem. Such as training data consisting of an input and expected output.

When compared to common machine learning methods, IP has several advantages. Since the hypothesis language used by inductive programming systems is often a functional or logic-based language. The language can be either domain specific or a Turing complete, general programming language. The complexity of the learned hypothesis can more easily be assessed. Being able to search for the simplest hypothesis which explains the provided examples means the amount of training data required is reduced. Additionally, while a learned hypothesis encoded in most machine learning models like neural networks is difficult or practically impossible to comprehend. The hypothesis' learned by inductive programming can more easily be analyzed and understood.

Understanding the hypothesis provides more options when validating the results. Machine learning techniques are often limited to using statistical methods to prove the accuracy and generalizability of a hypothesis[1]. When validating hypothesis' learned by IP, there is an additional option of inspecting the code, and sometimes parts of the hypothesis can be isolated and analyzed separately.

4.1 Types of inductive programming

Contemporary approaches can be classified into being either search based, analytical or a combination of the two.

4.1.1 Analytical approaches

Analytical approaches try to synthesize programs using training examples and domain specific background knowledge. These approaches are typically limited to solving problems which can be formulated as:

$$q^* = q \in Q : Correct(q) \tag{4.1}$$

with Q being the set of valid programs and $Correct : Q \longrightarrow \mathbb{B}$ being a predicate on the validity of the program q.

[1]With the exception of some methods like decision tree learning which provides interpretable trees.

Development of analytical approaches started in the 1970s with the work on systems like THESYS[57] which constructs recursive LISP programs from examples. These were improved in the 1990s with work mostly focused on inductive logic programming which uses logic languages as the hypothesis language. Some of the major work was done on systems like Golem [58], Progol [59], [60], and Dialogs [61]. The state-of-the-art analytical IP system is IGOR2, an improvement of the earlier IGOR system[62]–[64].

4.1.2 Search based approaches

Search-based approaches work by generating programs using some heuristic and evaluating them by testing how well they perform according to some measurements. These can usually be stated as a variant of the optimization problem:
Given a set of valid programs Q:

$$\min(pe_1(q), pe_2(q), \ldots, pe_k(q)) \tag{4.2}$$

with $q \in Q$ and pe_i being program evaluation functions used to balance multiple criteria such as program size, running time, and results from evaluation of training problems.

This process is typically more flexible than analytical induction because the solvable problems are not limited to being expressed by input/output examples, but also include problems where programs have to be evaluated as part of larger simulations. This flexibility does however come at the cost of being vastly more computationally expensive.

The difference between optimization based approaches is largely in how the search space is navigated. The following sections describe some of the most prominent methods.

MagicHaskeller

MagicHaskeller developed by Katayama [65] is one of the most known enumerative IP systems[66]. It is capable of synthesizing short, pure functional Haskell functions by exhaustively searching all possible programs, starting with short programs, and gradually increasing the length. Additionally, it has been explored using analytical synthesis to produce more promising programs based on techniques introduced in IGOR 2[65].

More recently, they have investigated whether the algorithm could be improved by collecting usage data for commonly used expressions from many synthesizes to bias the search towards more common constructs[67].

Genetic programming

Evolutionary computation has a long history with IP, and one such approach is Genetic Programming (GP) [68]. GP is similar to GA, but with the individuals represented by trees or some functional programming language of varying size. Like GA, evolution uses repeated applications the mechanisms: reproduction, selection, mutation, and recombination to evolve a population of programs towards better programs.

However, one of the most studied topics is the problem of GP producing bloated programs, with expressions growing indefinitely in size[69].

ADATE

The system used in this thesis, ADATE is an evolution inspired, local search algorithm. The system manages to naturally combine the benefits of evolutionary search, with a high bias towards smaller programs. A detailed description of ADATE is given in Section 4.3.

4.2 Applications

This section describes similar work where IP systems have been used to improve or design heuristics for algorithms.

4.2.1 Bent Function Synthesis

Hrbacek and Dvorak [70] won bronze medal in the 2014 Humies Awards[2] by using Cartesian Genetic Programming to synthesize bent functions. Bent functions are functions that are naturally hard to approximate and are therefore desirable in the field of cryptography.

4.2.2 Improving image analysis algorithms

There are multiple attempts at using both GP and ADATE to improve image segmentation algorithms. The most similar to the work in this thesis, is the work by Magnusson and Olsson [71]–[73] who used ADATE to improve the Canny edge detector algorithm.

4.2.3 Improving search heuristics

Both GP and ADATE have been used extensively to improve search algorithms. Many of these use GP to improve the heuristics used in varying forms of the knapsack problem[74]–[76] or scheduling problem [77]. More related work try to improve the local search heuristics for solving Boolean Satisfiability Problems[14], [78], [79].

4.3 ADATE

ADATE is an IP algorithm developed by Olsson [11] based on iterative deepening local search with evolution inspired heuristics. It is capable of generating functional programs with multiple, mutually recursive, auxiliary helper functions. The search algorithm evolves a kingdom of programs by systematically generating and testing a large amount of programs. The overall search process is outlined in Algorithm 4.

The search starts by initializing the kingdom K and inserting the initial program. The outer loop representing generations, iteratively deepends the W_{tot} variable, which controls how many children are to be produced for each parent program using the branching factor α. The smallest program which has not been expanded in the current generation is then selected for expansion, with the children inserted into the kingdom. This process is repeated until there are no more programs in the kingdom which has not been expanded in the current generation.

The program chosen for expansion is expanded for each of the forms discussed in Section 4.3.2, with the goal of producing W_{tot}/N_{forms} children for each form using iterative deepening of the $Cost\_limit$ variable using the branching factor β. The $Cost\_limit$ variable limits the size of each transformation, in effect defining the neighborhood of a program. The iterative deepening of $Cost\_limit$ is stopped when it reaches $20 \cdot W_{tot}/N_{forms}$

[2]For details see: `http://www.human-competitive.org/awards`

Algorithm 4 The overall ADATE search.

```
$i \leftarrow 0$
$K \leftarrow$ empty kingdom
INSERT( $K$, initial program )
repeat
    $W_{tot} = 100 \cdot \alpha^i$
    while $K$ contains programs not expanded with $W_{tot}$ do
        $q \leftarrow$ smallest program in $K$, not expanded with $W_{tot}$.
        for $n \leftarrow 1, N_{forms}$ do
            $c \leftarrow 0$
            $Cost\_limit \leftarrow 100$
            $j \leftarrow 0$
            while $Cost\_limit < \frac{20 \cdot W_{tot}}{N_{forms}} \wedge c < \frac{W_{tot}}{N_{forms}}$ do
                $Children \leftarrow$ EXPAND( p, $n$, $W_{tot} - c$, $Cost\_limit$ )    ▷ see Section 4.3.2
                $c \leftarrow c + |Children|$
                INSERT( $K$, $Children$ )
                $j \leftarrow j + 1$
                $Cost\_limit \leftarrow 100 \cdot \beta^j$
            end while
        end for
    end while
    $i \leftarrow i + 1$
until terminated by user
```

but can also end early when all programs have been expanded to produce W_{tot}/N_{forms} programs for each form.

This algorithm, when combined with the kingdom structure described in the next section allows ADATE to greedily search with a bias towards smaller programs while avoiding bloated programs and getting stuck in local optima.

4.3.1 The kingdom

The kingdom structure is the core of ADATE and has to balance many different requirements to maintain diversity. The solution is a kingdom structure inspired by Linnean Taxonomy. The kingdom consists of nine populations, one for each of the three evaluation functions: pe_1, pe_2 and pe_3 where programs are compared by lexical ordering using the criteria in Table 4.1. Each of these is evaluated using three different syntactic complexity measures. The populations are in turn organized into a grid with rows for spans of syntactic complexities and columns for spans of time complexities. Each cell in the grid is called a family and consists of a base individual and three classes. The classes are used to prevent ADATE from getting stuck in local optima due to missing links between the program and the initial program. In this thesis, only one column for time complexity is used.

When a program is considered inserted into the kingdom, it is, in turn, attempted inserted into all populations. A program is admitted to a family when it is better than all smaller programs. The program replaces the base individual in the family if it is not worse, and is smaller or has lower time complexity. If the program is unable to replace the

Table 4.1: Criteria for each evaluation functions. The functions are evaluated such that smaller of each criteria is better. N_c is number of correct problems, N_w is number of incorrect, G is the user defined grade, S is the syntactic complexity and T is the time complexity.

Evaluation function	**Criteria**				
	1	**2**	**3**	**4**	**5**
pe_1	$-N_c$	G	N_w	S	T
pe_2	$-N_c$	G	N_w	T	S
pe_3	N_w	$-N_c$	G	S	T
pe_{REQ}	$-N_c$	G	N_w		

base individual, the program is attempted inserted into one of the classes in the family. The program is rejected from the family if it is unable to enter any of the classes.

4.3.2 Expanding a program

ADATE attempts to create W programs using the process outlined in Algorithm 5. Fewer programs might be created if there are no possible transformation with the current *Cost_limit*, or if some the programs fail according to some criteria not accounted for during synthesis.

Algorithm 5 Expand program q to produce W children using form n with *Cost_limit*

```
procedure EXPAND( $q$, $n$, $W$, $Cost\_limit$ )
    $R \leftarrow \emptyset$
    for $i \leftarrow 1, W$ do
        $c \leftarrow$ SYNTHESIZECHILD( $q$, $n$, $Cost\_limit$ )
        if $c \neq null \wedge$ ISVALID( $c$ ) then
            $R \leftarrow R \cup \{c\}$
        end if
    end for
    return $R$
end procedure
```

Atomic transformations

Programs are transformed using one or more *compound transformations* which consist of one or more of the following *atomic transformations*:

***R*:** Replacement transformation replaces an expression with a synthesized expression. This is the only transformation which can change the semantics of the program.

***REQ*:** This transformation replaces an expression with another that have been shown empirically to not reduce the performance of the program according to pe_{REQ} defined in Table 4.1. This is used in compound transformations to reach a larger neighborhood of programs.

***CASE-DIST*:** Moves a case expression either out of, or into a function call. An example is shown in Figure 4.1.

***ABSTR*:** Abstracts an expression into a nested function with a maximum arity of 2. An example of this is shown in Figure 4.2.

***EMB*:** Changes either the argument, or return type of a nested function.

```
fun f x =
  case x =>
    true => g( 1 )
  | false => g( 2 )
```

(a) Before

```
fun f x =
  g( case x =>
    true => 1
  | false => 2 )
```

(b) After

Figure 4.1: Example of the CASE-DIST atomic transformation.

```
fun f x =
  1+2+3
```

(a) Before

```
fun f x =
   let
     g(V1, V2) => V1 + V2
   in
     g( 1, 2 ) + 3
   end
```

(b) After

Figure 4.2: Example of the ABSTR atomic transformation.

Compound transformations

Except for the first *atomic transformation*, the transformations have to be chosen according to the following coupling rules:

- REQ ⇒ R
- REQ ⇒ ABSTR
- ABSTR ⇒ R
- ABSTR ⇒ REQ
- ABSTR ⇒ REQ ⇒ REQ
- ABSTR ⇒ EMB
- CASE-DIST ⇒ ABSTR
- CASE-DIST ⇒ R
- EMB ⇒ R

Combining these rules in all possible ways result in $N_{forms} = 22$ forms such as: REQ ⇒ ABSTR ⇒ REQ ⇒ R.

4.3.3 Specification

The specification contains information about the problem to be solved by ADATE. It provides information about how a synthesized program is to be evaluated and which programs can be synthesized.

A specification is a text file consisting of two parts separated by a line containing only the text %%. The code above this line is written in ADATE-ML, a language which is described in Section 4.3.4. This part contains the definition of the f-function, supporting user defined data types and helper functions that are to be made available within f. The part below %% is code written in Standard ML, which contain everything else.

Grading programs

The grade is a user-defined measure of how well a program solves the problem. Grading is done by using two sets of problems: Training and Validation. When testing a program, all problems in these sets are passed the main function. In this thesis, this function implements a variation of the DE algorithm. The parts of this algorithm which are to be improved by ADATE is isolated into a function called f. This will be described more in the next section. The result from main is passed to a function output_eval_fun which evaluates the result to determine whether it should mark the problem as correctly solved and calculates the grade. The *grade* is used to provide a nuanced measure of how well the problem was solved.

In this project, the result is marked as not correct if it throws an exception at any point while being tested. This will often happen with invalid f-functions, which have invalid raise statements. In addition, exceptions are also thrown to quit the testing of programs which returns other types of bad results. Examples include real values outside the accepted range. The grade for the problems where this occur are set to the value 10^{14}, which is much larger than any valid grade and would cause the program to be discarded by ADATE.

The f-function

The f-function encapsulate the parts of an algorithm which is to be improved by ADATE written in the ADATE-ML language which is described in Section 4.3.4. In addition to only using the functionality provided by ADATE-ML, the function also must take only a single argument. Passing multiple arguments can be done using user-defined algebraic data types. The code must also be side-effect free, which preclude modifying arrays or other mutable data structures in helper functions.

In this project, it was necessary to improve several parts of the algorithms simultaneously. This was solved by using a user-defined data type for the argument with constructors for each of the parts to be improved.

Controlling program synthesis

ADATE provides several methods for controlling which programs can be synthesized to reduce the size of the search space. The two methods used in this project are max_syntactic_complexity which is used to prevent ADATE from producing too large programs, and Funs_to_use which is a list of functions ADATE is allowed to use in synthesized programs.

The number of allowed functions is typically kept as small as possible to limit the branching necessary to search through improvements. The exact content of the list is problem specific. For problems dealing with real numbers, it is practical to include basic arithmetic operators like addition and division.

4.3.4 ADATE-ML

ADATE-ML is a stripped-down version of Standard ML with many of the redundant features removed. This simple language makes the ADATE-ML compiler much smaller and quicker than compilers for most other languages.

The set of features left in the language is designed to match with the features used by ADATE during program synthesis. Having to handle multiple ways of achieving the same task, such as declaring a variable, would cause an unnecessarily large search space.

Some notable removed features are: if expressions, selectors, boolean operators, variable deceleration using let-statements, and pattern matching in function decelerations as these can all be achieved using case expressions. Similarly, polymorphic types are also removed. Let statements is allowed, but can only contain a single function declaration.

The built-in library only contains basic data types like bool, real, and array, in addition to basic operations on these types.

The features allowed within the f-function is even more scarce. All code within the f-function has to be purely functional which prevents using mutable operations like updates on arrays if that changes the behavior of the program. Additionally, real numbers have to be defined within f using the rconst constructor, but this is omitted in the code listings in this project as it has no practical effect outside the ADATE search.

Chapter 5

Improving Differential Evolution

The standard DE has been implemented in many machine learning and numerical optimization libraries such as SciPy[1] and LabView[2]. Since the mutation operator is configurable in these tools, significant improvements could quickly be adopted and used by a larger group of people.

This chapter aims to understand how synthesized versions of DE can be evaluated in ADATE and which parts should be improved by attempting to enhance the DE/rand/1 mutation operator in the standard DE algorithm.

Two experiments are used. The first requires ADATE to find an improved mutation operator using knowledge about the best individual in the population and a few randomly selected individuals. Following a preliminary evaluation, a second experiment based on the same framework but with the population variance made available for ADATE was designed.

This chapter is organized as follows: Section 5.1 provides details on the experiment, Section 5.2 reports on and compares the performance of the improved mutation operator with other mutation operators, and Section 5.3 provides a summary of this chapter.

5.1 Experiment

Improving an algorithm using ADATE requires a starting individual that is to be improved, and a method for comparing the performance of synthesized programs. Here, this is provided by isolating the code responsible for generating a trial vector $\mathbf{v_i^{(t)}}$ into an isolated f-function, as described in Section 5.1.2, and evaluating the performance of a program, as outlined in Section 5.1.4.

5.1.1 Which part to improve?

The standard DE algorithm has three primary candidates for improvement, specifically, the crossover, the selection, and the mutation operator.

As discussed in Section 3.3, the crossover has a large effect on the search and is of particular importance when optimizing non-separable functions. However, the heuristics

[1]`https://docs.scipy.org/doc/scipy-0.18.1/reference/generated/scipy.optimize.differential_evolution.html`

[2]`http://zone.ni.com/reference/en-XX/help/371361H-01/gmath/global_optimization/`

are encoded in the structure and enumeration of lists, making incremental refinements difficult.

The selection has been unchanged from the original publication by Storn and Price [7], to state-of-the-art variants of DE, making it less likely that improvements could significantly enhance the performance. Additionally, any improvement would also require larger structural changes to the code.

The mutation operator has been among the most studied parts of the algorithm with numerous variations published over the years as discussed in Section 3.2. Additionally, the number of transformations needed to change one into another would mostly, if ADATE were provided all the required background knowledge about the population, be just a matter of changing some variables or adding some terms to a linear equation. All this makes mutation the best step to improve using ADATE.

The question then becomes which mutation operator to improve. Except for the Trigonometric mutation operator [29], all mutation operators discussed in Section 3.2 have a similar structure. The DE/rand/1 operator is chosen because it is likely to be the most popular. The performance of the operator has also been extensively evaluated, which provides a measure the implementation can be compared with to ensure correct implementation[80].

5.1.2 The f-function

The DE/rand/1 operator is described by Equation (3.3), repeated here for convenience:

$$\mathbf{v_i^{(t)}} = \mathbf{x_{R_1}^{(t)}} + F\left(\mathbf{x_{R_2}^{(t)}} - \mathbf{x_{R_3}^{(t)}}\right).$$

Here $\mathbf{v_i^{(t)}}$ is the mutated donor vector for individual i at time t, and F is a scaling factor controlling the step size. The vectors $\mathbf{x_{R_1}^{(t)}}$, $\mathbf{x_{R_2}^{(t)}}$ and $\mathbf{x_{R_3}^{(t)}}$ are three randomly chosen individuals from the population.

To create the f-function, DE/rand/1 is implemented in ADATE-ML as shown in Listing 5.1 with a scaling factor $F = 0.28$, a value found by using a simple grid search on the training problems.

Listing 5.1: Original formulation of mutation in DE to be improved by ADATE. For readability, the rconst wrapper round real numbers are removed.

```
datatype real_list = rnil | rcons of real * real_list
datatype real_list_list = rlnil | rlcons of real_list * real_list_list

fun f( Population , Best , Z1, Z2 ) =
let
  fun build( Columns ) =
    case Columns of
      rlnil => rnil
    | rlcons( Col, RestColumns ) =>
      rcons(
        case Col of
          rnil => raise NA1
        | rcons( X1, X1Rest ) =>
          case X1Rest of
            rnil => raise NA2
          | rcons( X2, X2Rest ) =>
            case X2Rest of
```

```
                rnil => raise NA3
              | rcons( X3, X3Rest ) =>
                X1 + 0.28 * ( X2 - X3 )
          , build RestColumns )
in
  build( Population )
end
```

The Population argument is of type real_list_list which stores each column as a real_list. The best individual is stored in the variable Best of type real_list. Z1 and Z2 are random, real values drawn from a uniform distribution between 0 and 1.

All X values are normalized to the range -0.5 to 0.5 before f is called using the formula:

$$\phi(\mathbf{x}) = (\mathbf{x} - \mathbf{x_{min}}) \,./\, (\mathbf{x_{max}} - \mathbf{x_{min}}) - 0.5. \tag{5.1}$$

Here, ./ denote element-wise division.

Normalization prevents overfitting to the search space of the problems used by ADATE. It can also have a beneficial effect on the ADATE search, because constants introduced as part of program synthesis are randomly drawn from a distribution that produces constants from the same range more frequently. Additionally, this range makes it easier to introduce nonlinearity, using the tanh operator. The normalization itself does not significantly affect the performance of the algorithm. The return vector from the f-function is similarly denormalized before used further.

The implementation was verified by testing it on the CEC 2013 benchmark problems[81], and comparing the performance against the results published by Qin and Li [80].

Variance information

For the second experiment, the type of Population was changed to

Listing 5.2: Datatype for population with variance

```
datatype population = popnil | popcons of ( real * real_list ) * population
```

Here the standard deviation of the column is returned in the first argument of popcons, while the list of all values from the column and the remaining parts of the population is in the second.

Additionally, the argument Z2 to the f-function was changed to be drawn from a normal distribution with $\sigma = 1$ centered around $\mu = 0$ to make it easier for ADATE to find useful distributions of random numbers.

5.1.3 Allowed functions

The list of allowed functions for this experiment is primarily motivated by what would be necessary for ADATE to transform any of the mutation operators discussed in Section 3.2 into any of the others, if provided with the necessary information about the population.

The transformation between most of the operators can be accomplished using only $+$, $-$, and $*$ on real numbers. To create the Trigonometric operator, ADATE would also need to use / and compare two real numbers using the $<$ operator. To allow ADATE to create interesting non-linear calculations, the tanh operator is also included.

To be able to change the structure of the algorithm, ADATE must also be allowed to use constructors for the algebraic types used within f.

5.1.4 Grading synthesized programs

Three options have been considered for grading the synthesized programs produced by ADATE, namely, designing problems based on practical machine learning models, landscape generators and synthetic benchmark functions designed to test optimization algorithms.

Designing problems based on practical machine learning models is attractive because a large number of problems could be created, and many models such as feedforward neural networks can be evaluated quickly. However, it would be prohibitively time-consuming and difficult to ensure that the fitness landscapes are representative of most problems encountered for general optimization tasks.

Gallagher and Yuan [82] have proposed a landscape generator based on Gaussian mixture models capable of generating an unlimited number of optimization problems of varying complexity. However, it is unlikely to produce an equal amount of all types of landscapes, and complicated problems become computationally expensive to calculate compared to the other options.

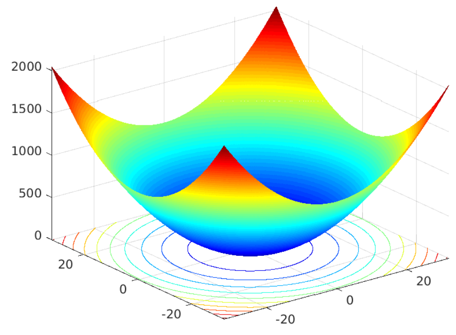
Because of the limitations above, synthetic functions designed to test optimization problems were used to grade synthesized programs. While only a limited quantity of these problems exist, they were designed to study optimization, and as a result, it exists problems for most of the important types of fitness landscapes.

The functions were selected such that the original DE algorithm were unable to solve the problems within the constraints on the number of generations and parameter configuration. However, it could solve some of the simpler problems with a larger number of generations, or by using parameters optimized for each problem.

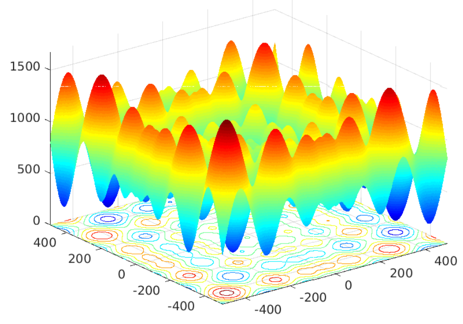
Training

Five problems, listed in Table 5.1, were used for the training set. These were selected to represent several types of search landscapes and include both the simple Sphere function (see Figure 5.1a) and the difficult Schwefel function (see Figure 5.1b).

These problems are all of 20 dimensions and optimized using a population size of 20 individuals.



(a) Sphere function



(b) Schwefel function

Figure 5.1: Plots of two functions used for grading synthesized programs when improving DE.

Table 5.1: Problems in the training set used by ADATE. The definition for each function is available in Appendix A.

f	**Name**	t_{max}	$x_i \in$
f_1	Ackleys Function	200	$[-32, 32]$
f_{41}	Rastrigin function	200	$[-32, 32]$
f_{53}	Sphere function	150	$[-10, 10]$
f_{22}	Griewank	150	$[-32, 32]$
f_{46}	Schwefel	250	$[-500, 500]$

Validation

The best performing programs found by ADATE on the training set were validated using a significantly more extensive set of problems. The 45 problems used for validation are listed in Table 5.2. These include problems of dimensionality 40, 30 and 2 with population sizes ranging from 30 to 75.

Randomization

Most of the problems have a global minimum at $\mathbf{x} = 0$, something which ADATE would quickly exploit. Additionally, many of the problems are symmetric around the minimum. To prevent overfitting to functions with these properties, the functions are shifted by a random amount.

The random shift vector for each problem was calculated as follows:

$$s_j = x_{min,j} + rand_j(0,1)(x_{max,j} - x_{min,j}) \tag{5.2}$$

The function was then shifted by:

$$x''_j = \min(\max(x_j + s_j, x_{min,j}), x_{max,j}) \tag{5.3}$$

To further prevent overfitting to the seeds, each program is evaluated using 11 different seeds, each calculating a unique shift. The final grade becomes the median of the tries on each problem.

Ensuring problem equality

All problems used in the specification should contribute equally to the grade. However, a simple summation of the optimization results would cause a subset of the problems to dominate, because many of the problems have different global optima and ranges. This was solved by measuring how much the performance on the problem has improved.

First, the median of the best fitness values found by the original DE using 11 different seeds, designated here for problem p as $\tilde{f}_{default,p}$, is calculated prior to running ADATE. The grade for a program is then calculated as the ratio of the achieved error to the median of the error reached by the original algorithm as follows:

$$\text{grade} = \sum_p \tanh\left(\frac{f_{best,p}}{\tilde{f}_{default,p}}\right) \tag{5.4}$$

Here $f_{best,p}$ is the fitness of the best individual in the population when the search is terminated. The tanh operator limits how much each problem can impact the grade.

Table 5.2: Problems used for validation in ADATE when improving DE. The definition for each function is available in Appendix A.

f	**Name**	**Dim**	t_{max}	N_p	$x_i \in$
f_1	Ackleys Function	40	400	30	$[-32, 32]$
f_6	Bent cigar	40	500	30	$[-100, 100]$
f_{16}	Discus Function	40	500	30	$[-100, 100]$
f_{22}	Griewank	40	400	50	$[-32, 32]$
f_{23}	HappyCat Function	40	500	30	$[-100, 100]$
f_{24}	HGBat Function	40	500	30	$[-100, 100]$
f_{25}	High Conditioned Elliptic	40	500	30	$[-100, 100]$
f_{27}	Katsuura Function	40	250	30	$[-100, 100]$
f_{33}	Modified Schwefel function	40	500	30	$[-100, 100]$
f_{37}	Powell sum	40	70	30	$[-1, 1]$
f_{39}	Qing's Function	40	500	30	$[-500, 500]$
f_{41}	Rastrigin function	40	400	50	$[-32, 32]$
f_{46}	Schwefel	40	600	75	$[-500, 500]$
f_{47}	Schwefel F2.21	40	500	30	$[-500, 500]$
f_{53}	Sphere function	40	400	30	$[-32, 32]$
f_{53}	Sphere function	40	500	75	$[-10, 10]$
f_{54}	Step Function No.02	40	500	30	$[-100, 100]$
f_{61}	Weierstrass	40	500	30	$[-100, 100]$
f_{62}	Whitley	40	500	30	$[-10.24, 10.24]$
f_{64}	Xin-She Yangs Function No.02	40	500	30	$[-2\pi, 2\pi]$
f_{67}	Expanded Griewanks, Rosenbrocks	40	500	30	$[-100, 100]$
f_{68}	Expanded Scaffers F6 Function	40	500	30	$[-100, 100]$
f_1	Ackleys Function	30	500	50	$[-32, 32]$
f_6	Bent cigar	30	500	30	$[-100, 100]$
f_{16}	Discus Function	30	500	30	$[-100, 100]$
f_{22}	Griewank	30	500	50	$[-32, 32]$
f_{23}	HappyCat Function	30	500	30	$[-100, 100]$
f_{24}	HGBat Function	30	500	30	$[-100, 100]$
f_{25}	High Conditioned Elliptic	30	500	30	$[-100, 100]$
f_{27}	Katsuura Function	30	200	20	$[-100, 100]$
f_{33}	Modified Schwefel function	30	500	30	$[-100, 100]$
f_{37}	Powell sum	30	70	30	$[-1, 1]$
f_{39}	Qing's Function	30	500	30	$[-500, 500]$
f_{41}	Rastrigin function	30	500	50	$[-32, 32]$
f_{46}	Schwefel	30	600	75	$[-500, 500]$
f_{47}	Schwefel F2.21	30	500	30	$[-500, 500]$
f_{53}	Sphere function	30	500	50	$[-32, 32]$
f_{53}	Sphere function	30	500	75	$[-10, 10]$
f_{54}	Step Function No.02	30	500	30	$[-100, 100]$
f_{61}	Weierstrass	30	500	20	$[-100, 100]$
f_{62}	Whitley	30	500	30	$[-10.24, 10.24]$
f_{64}	Xin-She Yangs Function No.02	30	500	30	$[-2\pi, 2\pi]$
f_{67}	Exp. Griewanks, Rosenbrocks	30	500	30	$[-100, 100]$
f_{68}	Expanded Scaffers F6 Function	30	500	30	$[-100, 100]$
f_{10}	Bukin's Function No.06	2	500	30	$x_1 \in [-15, -5]$ $x_2 \in [-3, 3]$

5.2 Results

5.2.1 Improved mutation operator

The first improved mutation operator found by ADATE, referred to here as DE/ADATE1 can be expressed as:

$$v_{i,j}^{(t)} = F\left(x_{R_2,i,j}^{(t)} - x_{R_3,i,j}^{(t)}\right) + \begin{cases} x_{R_3,i,j}^{(t)} & \text{if } x_{R_1,i,j}^{(t)} < 0.050668748 \\ x_{R_2,i,j}^{(t)} & \text{Otherwise} \end{cases} \tag{5.5}$$

with F being the difference between two random numbers drawn from a uniform distribution between 0 and 1.

Except for adding a condition that switches the base individual if $x_{R_1,i,j}^{(t)} < 0.050668748$, the mutation operator is similar to the original. The condition can act as a primitive measure of the population variance for problems with an optima sufficiently far from $x_j = 0.050668748$, because the condition will gradually stabilize to either $x_{R_2,i,j}^{(t)}$ or $x_{R_3,i,j}^{(t)}$ when the individuals become localized to either side of 0.050668748.

Switching the base individual has the effect of introducing some limited independence between the components early in the search. The other significant modification of the original is replacing the constant F value used in DE/Rand/1 with the difference between two uniform random numbers, which results in a triangular distribution centered around 0.

5.2.2 Improved mutation operator with variance

The second improved mutation operator, based on the experiment with the component standard deviation available, results in the following operator referred to here as DE/ADATE2:

$$v_{i,j}^{(t)} = x_{R_2,i,j}^{(t)} + \tanh\left(\tanh\left(\tanh\left(rand_{i,j}(0,1) - 0.28 - \sigma_j^{(t)}\right)\left(x_{R_2,i,j}^{(t)} - x_{R_1,i,j}^{(t)}\right)\right)\right). \tag{5.6}$$

Here $\sigma_j^{(t)}$ is the standard deviation for component j across all individuals in the population at time t.

5.2.3 Analysis and performance

The generalizing performance of the algorithm was tested on the CEC 2014 dataset using 10 and 30 dimensions, and compared against the mutation operators described in Section 3.2. To achieve optimal performance, the parameters for each mutation operator were tuned, using the sum of the medians for each function resulting from 51 trials on each function. The optimal parameters for each mutation operator for problems of dimensions 10 and 30, are listed in Tables 5.3 and 5.4, respectively. The performance of each mutation operator on all functions is listed in Tables 5.5 to 5.8.

The performance of the mutation operators found by ADATE is worse than most of the other mutation operators on the majority of the functions. The reason for this can be seen in Figure 5.2. While the improved mutation operators achieve better results for a few functions, on most, they quickly start performing worse than the original DE/rand/1 as the available number of function evaluations is increased. This is because the improved mutation operator has specialized to the low number of generations used in the training and validation problems.

Table 5.3: Best performing parameters across all CEC 2014 benchmark functions on 10 dimensions, as measured by the sum of the medians. The parameters were found by testing $Np \in \{20, 40, 45, 50, 55, 60, 80, 100, 150\}$, $Cr \in \{0.1, 0.5, 0.9\}$, and $F \in \{0.5, 0.8\}$.

Mutation operator	**Np**	**Cr**	**F**
DE/Rand/1	60	0.9	0.8
DE/Best/1	55	0.9	0.8
DE/current-to-best/1	100	0.5	0.5
DE/Rand/2	50	0.9	0.5
DE/Best/2	20	0.9	0.8
DE/Randrl	80	0.9	NA
DE/ADATE1	80	0.5	NA
DE/ADATE2	55	0.5	NA

Table 5.4: Best performing parameters across all CEC 2014 benchmark functions on 30 dimensions, as measured by the sum of the medians. The parameters were found by testing $Np \in \{20, 40, 45, 50, 55, 60, 80, 100, 150\}$, $Cr \in \{0.1, 0.5, 0.9\}$, and $F \in \{0.5, 0.8\}$.

Mutation operator	**Np**	**Cr**	**F**
DE/Rand/1	80	0.9	0.5
DE/Best/1	100	0.9	0.5
DE/current-to-best/1	20	0.9	0.8
DE/Rand/2	20	0.9	0.5
DE/Best/2	20	0.9	0.5
DE/Randrl	40	0.9	NA
DE/ADATE1	50	0.9	NA
DE/ADATE2	45	0.9	NA

Table 5.5: Performance of mutation operators on CEC 2014 benchmark functions 1 to 15 with 10 dimensions using the parameters performing best across all CEC 2014 functions as measured by the sum of all medians.

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
DE/Rand/1															
Best	2.77e−3	1.04e−7	1.42e−12	1.47e−8	2.01e1	2.92e−3	8.14e−2	1.81	2.74	1.56e1	8.17e1	1.04e−1	4.63e−2	1.01e−1	8.53e−1
Worst	7.44e−2	8.26e−6	1.01e−10	3.48e1	2.03e1	6.78	6.34e−1	2.78e1	4.33e1	1.58e2	1.04e3	8.56e−1	3.58e−1	2.52e−1	3.77
Median	1.32e−2	5.91e−7	1.27e−11	4.34	2.02e1	9.14e−1	4.60e−1	1.94e1	3.23e1	5.41e1	2.40e2	2.32e−1	2.52e−1	1.88e−1	2.68
Mean	1.69e−2	9.23e−7	1.85e−11	1.37e1	2.02e1	1.29	4.31e−1	1.75e1	2.84e1	6.31e1	2.79e2	2.68e−1	2.36e−1	1.83e−1	2.53
STD	1.31e−2	1.23e−6	1.85e−11	1.65e1	5.30e−2	1.24	1.67e−1	7.01	1.04e1	3.23e1	2.01e2	1.32e−1	6.76e−2	3.53e−2	6.91e−1
DE/Best/1															
Best	1.21e−3	3.23e−8	5.12e−13	7.49e−9	2.00e1	4.16e−3	3.81e−2	1.39	1.01	2.76e1	7.97e1	9.15e−2	3.88e−2	9.22e−2	4.78e−1
Worst	2.65e−2	1.20e−6	2.43e−11	3.48e1	2.04e1	4.17	6.18e−1	2.89e1	3.97e1	1.35e2	1.21e3	6.94e−1	3.23e−1	2.66e−1	3.62
Median	8.05e−3	1.71e−7	3.47e−12	4.34	2.02e1	9.07e−1	4.51e−1	1.60e1	2.71e1	6.13e1	2.91e2	2.31e−1	2.29e−1	1.91e−1	2.50
Mean	9.39e−3	2.60e−7	5.27e−12	1.37e1	2.02e1	1.20	3.89e−1	1.53e1	2.32e1	6.31e1	3.50e2	2.51e−1	2.06e−1	1.85e−1	2.41
STD	6.13e−3	2.32e−7	5.19e−12	1.65e1	8.15e−2	1.12	1.84e−1	6.63	1.15e1	2.67e1	2.69e2	1.37e−1	7.34e−2	4.39e−2	8.15e−1
DE/current-to-best/1															
Best	9.35	0.00	0.00	8.18e−3	1.02e−4	0.00	2.98e−11	0.00	5.87	9.67e−2	1.86e2	3.63e−1	6.41e−2	6.57e−2	9.13e−1
Worst	1.12e3	0.00	0.00	3.48e1	2.03e1	2.79	1.71e−1	9.95e−1	1.43e1	2.19e2	9.70e2	8.96e−1	1.75e−1	2.93e−1	2.03
Median	5.41e1	0.00	0.00	3.48e1	2.02e1	0.00	3.97e−2	0.00	9.89	3.04e1	6.78e2	6.38e−1	1.21e−1	1.36e−1	1.45
Mean	1.15e2	0.00	0.00	3.04e1	1.90e1	3.09e−1	4.64e−2	3.90e−2	9.90	5.35e1	6.68e2	6.22e−1	1.20e−1	1.41e−1	1.49
STD	1.85e2	0.00	0.00	1.12e1	4.07	5.74e−1	3.74e−2	1.95e−1	2.09	5.25e1	1.71e2	1.35e−1	2.71e−2	5.39e−2	2.61e−1
DE/rand/2															
Best	2.75	2.25	2.52e−1	2.88e−1	1.82e−1	5.85e−2	5.39e−1	3.29e2	1.22e2	1.36e2	1.00e2	1.97	3.57e2	1.32e2	4.62e2
Worst	3.52	1.14e2	4.39	1.31	1.10	5.05	1.97e1	3.29e2	1.45e2	2.01e2	1.00e2	4.00e2	4.78e2	2.23e2	5.01e2
Median	3.18	3.10e1	1.89	7.68e−1	6.82e−1	4.35e−1	3.29	3.29e2	1.38e2	2.01e2	1.00e2	3.34	3.69e2	2.22e2	4.63e2
Mean	3.15	4.18e1	1.98	7.72e−1	6.47e−1	5.57e−1	6.58	3.29e2	1.37e2	1.87e2	1.00e2	1.70e2	3.77e2	2.20e2	4.65e2
STD	1.77e−1	3.05e1	9.59e−1	2.52e−1	1.86e−1	7.01e−1	6.35	5.74e−14	4.41	2.54e1	3.15e−2	1.76e2	2.42e1	1.27e1	8.03
DE/best/2															
Best	3.61e−1	4.89e−4	2.22e−9	4.80e−6	2.01e1	3.36e−1	4.83e−1	1.93e1	2.76e1	1.19e2	9.57e1	8.41e−2	1.83e−1	9.74e−2	2.03
Worst	7.43e4	1.05e−1	3.54e−6	3.48e1	2.05e1	9.07	8.18e−1	3.99e1	5.47e1	6.59e2	1.44e3	1.11	4.82e−1	3.07e−1	5.10
Median	2.45	1.27e−2	1.89e−7	4.34	2.02e1	3.54	6.77e−1	3.24e1	4.22e1	3.15e2	5.67e2	2.34e−1	3.53e−1	2.33e−1	3.77
Mean	1.46e3	2.05e−2	4.46e−7	1.13e1	2.02e1	4.06	6.83e−1	3.16e1	4.15e1	3.40e2	7.19e2	3.42e−1	3.43e−1	2.32e−1	3.70
STD	1.04e4	2.25e−2	6.69e−7	1.53e1	9.53e−2	2.20	7.68e−2	4.66	6.48	1.30e2	4.35e2	2.44e−1	7.40e−2	4.45e−2	6.88e−1
DE/Randrl															
Best	0.00	0.00	0.00	0.00	2.00e1	0.00	2.46e−2	9.83e−5	5.97	4.37e−1	4.02e1	5.53e−3	1.12e−1	6.44e−2	1.32
Worst	1.42e−14	0.00	0.00	3.48e1	2.04e1	3.44	5.06e−1	2.38e1	3.88e1	1.35e2	1.15e3	3.67e−1	2.23e−1	2.15e−1	3.06
Median	0.00	0.00	0.00	3.48e1	2.02e1	8.95e−1	3.41e−1	1.76e1	2.71e1	4.90e1	2.05e2	1.38e−1	1.58e−1	1.37e−1	2.32
Mean	1.67e−15	0.00	0.00	2.43e1	2.02e1	1.12	2.69e−1	1.74e1	2.56e1	5.16e1	3.50e2	1.47e−1	1.62e−1	1.40e−1	2.28
STD	4.62e−15	0.00	0.00	1.57e1	8.43e−2	9.78e−1	1.71e−1	3.97	6.71	2.17e1	3.27e2	8.82e−2	2.85e−2	3.04e−2	3.71e−1
DE/ADATE1															
Best	9.64e2	1.28e−2	2.86	1.49e−1	2.01e1	0.00	0.00	0.00	0.00	3.54	2.97e2	3.72e−1	6.91e−2	1.42e−1	9.97e−1
Worst	2.01e6	7.40e3	1.26e3	3.48e1	2.03e1	8.95e−1	3.69e−2	9.95e−1	1.68e1	6.41e1	1.08e3	8.95e−1	2.12e−1	3.71e−1	1.79
Median	2.22e4	4.18e2	1.22e2	3.48e1	2.02e1	0.00	1.14e−13	0.00	1.08e1	3.54e1	7.40e2	6.73e−1	1.32e−1	2.70e−1	1.46
Mean	1.02e5	1.19e3	1.99e2	3.33e1	2.02e1	1.75e−2	6.06e−3	1.17e−1	1.05e1	3.31e1	7.24e2	6.70e−1	1.29e−1	2.70e−1	1.43
STD	2.99e5	1.64e3	2.46e2	6.80	5.50e−2	1.25e−1	9.16e−3	3.24e−1	3.48	1.35e1	1.45e2	1.17e−1	3.11e−2	6.52e−2	1.90e−1
DE/ADATE2															
Best	1.33e3	1.42	4.42	9.90e−2	1.80e1	0.00	0.00	0.00	9.95e−1	3.48	8.26	3.96e−1	7.52e−2	1.18e−1	1.04
Worst	4.66e5	7.12e3	3.04e3	3.48e1	2.03e1	2.89e−2	3.45e−2	2.98	1.39e1	1.55e2	8.80e2	1.00	1.86e−1	3.62e−1	2.20
Median	1.98e4	5.02e2	9.35e1	3.48e1	2.02e1	0.00	1.03e−2	9.95e−1	6.03	2.38e1	5.37e2	6.96e−1	1.26e−1	2.12e−1	1.52
Mean	6.96e4	1.12e3	3.00e2	2.73e1	2.01e1	1.18e−3	1.06e−2	7.41e−1	6.44	2.73e1	4.92e2	6.86e−1	1.26e−1	2.20e−1	1.53
STD	1.10e5	1.45e3	4.92e2	1.43e1	4.17e−1	5.67e−3	8.30e−3	7.92e−1	3.62	2.24e1	1.85e2	1.30e−1	2.77e−2	6.06e−2	2.17e−1

Table 5.6: Performance of mutation operators on CEC 2014 benchmark functions 16 to 30 with 10 dimensions using the parameters performing best across all CEC 2014 functions as measured by the sum of all medians.

	16	**17**	**18**	**19**	**20**	**21**	**22**	**23**	**24**	**25**	**26**	**27**	**28**	**29**	**30**
DE/Rand/1															
Best	9.84e−1	2.11e−1	6.63e−2	1.93e−1	6.75e−2	5.36e−2	1.11e−1	3.29e2	1.09e2	1.31e2	1.00e2	1.98	3.69e2	2.22e2	4.62e2
Worst	3.45	9.46e1	3.44	1.55	1.84	8.50e−1	2.33e1	3.29e2	1.48e2	2.01e2	1.00e2	4.00e2	4.97e2	2.27e2	5.13e2
Median	3.01	4.88	7.68e−1	8.70e−1	8.11e−1	3.78e−1	4.71	3.29e2	1.34e2	2.01e2	1.00e2	4.08	3.79e2	2.23e2	4.63e2
Mean	2.79	8.71	9.21e−1	8.77e−1	7.87e−1	3.77e−1	8.04	3.29e2	1.30e2	1.92e2	1.00e2	1.49e2	3.82e2	2.23e2	4.69e2
STD	6.06e−1	1.39e1	7.39e−1	3.38e−1	3.08e−1	2.18e−1	8.92	5.74e−14	1.15e1	2.19e1	4.47e−2	1.85e2	2.14e1	8.18e−1	1.26e1
DE/Best/1															
Best	7.17e−1	3.18e−1	4.58e−2	9.63e−2	1.28e−2	3.25e−2	9.55e−2	3.29e2	1.07e2	1.42e2	1.00e2	1.40	3.57e2	1.01e2	4.63e2
Worst	3.54	4.29e1	7.21	1.62	1.07	1.68e1	2.32e1	3.29e2	1.46e2	2.01e2	1.00e2	4.00e2	4.11e2	2.27e2	5.01e2
Median	2.83	5.00	5.16e−1	7.75e−1	7.20e−1	3.40e−1	3.71	3.29e2	1.27e2	2.01e2	1.00e2	4.01	3.79e2	2.23e2	4.63e2
Mean	2.61	7.05	1.03	7.89e−1	6.37e−1	8.28e−1	8.04	3.29e2	1.27e2	1.99e2	1.00e2	1.50e2	3.79e2	2.19e2	4.67e2
STD	6.10e−1	8.69	1.16	3.12e−1	2.62e−1	2.49	9.16	5.74e−14	1.15e1	9.07	6.38e−2	1.68e2	1.14e1	2.11e1	9.94
DE/current-to-best/1															
Best	1.53	1.09e1	2.63e−1	8.37e−3	2.02e−3	1.09e−1	3.95e−2	3.29e2	1.01e2	1.06e2	1.00e2	1.15	3.57e2	2.08e2	4.60e2
Worst	2.77	3.62e2	2.71e1	2.50	9.16	1.25e2	3.71e1	3.29e2	1.22e2	2.01e2	1.00e2	4.00e2	4.98e2	1.96e6	8.29e2
Median	2.13	7.76e1	2.14	9.99e−1	4.40e−1	7.32e−1	2.04e1	3.29e2	1.14e2	1.36e2	1.00e2	2.97	4.56e2	2.27e2	4.91e2
Mean	2.15	1.11e2	3.49	8.28e−1	7.91e−1	1.81e1	1.57e1	3.29e2	1.14e2	1.53e2	1.00e2	1.76e2	4.29e2	3.87e4	5.30e2
STD	2.57e−1	7.83e1	4.44	6.09e−1	1.42	3.44e1	9.40	5.74e−14	4.20	3.61e1	2.49e−2	1.94e2	5.34e1	2.75e5	8.20e1
DE/rand/2															
Best	7.90e−12	0.00	0.00	0.00	2.01e1	2.75e−6	3.48e−1	1.47e1	1.85e1	1.57e2	1.93e2	1.34e−1	9.51e−2	9.46e−2	1.69
Worst	1.21e−9	0.00	0.00	3.48e1	2.04e1	5.12	6.82e−1	2.85e1	4.09e1	6.69e2	1.47e3	1.32	2.86e−1	2.40e−1	3.18
Median	6.81e−11	0.00	0.00	0.00	2.02e1	8.95e−1	5.25e−1	2.17e1	3.18e1	3.45e2	7.96e2	9.59e−1	1.94e−1	1.62e−1	2.55
Mean	1.36e−10	0.00	0.00	1.10e1	2.02e1	1.21	5.09e−1	2.20e1	3.01e1	3.48e2	8.60e2	9.07e−1	1.92e−1	1.62e−1	2.49
STD	1.94e−10	0.00	0.00	1.56e1	9.16e−2	1.18	7.53e−2	3.23	4.61	1.09e2	3.97e2	2.71e−1	3.98e−2	3.06e−2	4.07e−1
DE/best/2															
Best	3.08	1.48e1	4.75	6.93e−1	1.17	7.55e−1	1.10	3.29e2	1.26e2	1.54e2	1.00e2	3.60	3.57e2	2.11e2	4.60e2
Worst	3.71	2.68e2	1.37e1	2.61	8.31	3.54e6	3.84e1	3.29e2	1.62e2	2.02e2	1.00e2	4.46e2	5.16e2	2.05e6	5.92e2
Median	3.37	9.38e1	8.42	1.72	4.42	4.23	2.35e1	3.29e2	1.48e2	2.01e2	1.00e2	3.04e2	3.87e2	2.24e2	4.74e2
Mean	3.38	9.72e1	8.75	1.73	4.40	9.29e4	2.09e1	3.29e2	1.47e2	2.00e2	1.00e2	1.95e2	4.03e2	1.45e5	4.83e2
STD	1.78e−1	6.58e1	2.34	3.34e−1	1.78	5.20e5	1.09e1	5.74e−14	8.50	7.58	4.93e−2	1.87e2	3.83e1	4.56e5	2.47e1
DE/Randrl															
Best	1.01	2.27e−13	1.62e−3	1.97e−2	4.13e−5	3.72e−5	8.04e−5	3.29e2	1.09e2	1.12e2	1.00e2	1.15	3.69e2	2.22e2	4.62e2
Worst	3.35	1.20e2	1.34	1.19	6.94e−1	1.12	6.31	3.29e2	1.41e2	2.01e2	1.00e2	4.00e2	3.89e2	3.32e2	4.99e2
Median	2.98	1.41	2.44e−1	5.61e−1	1.78e−2	3.45e−1	3.13e−1	3.29e2	1.32e2	2.01e2	1.00e2	2.54	3.75e2	2.22e2	4.62e2
Mean	2.94	1.04e1	4.10e−1	5.57e−1	1.20e−1	4.52e−1	4.11e−1	3.29e2	1.31e2	1.95e2	1.00e2	1.70e2	3.75e2	2.24e2	4.64e2
STD	3.75e−1	1.91e1	4.28e−1	2.51e−1	1.91e−1	3.29e−1	9.25e−1	5.74e−14	6.66	1.99e1	2.38e−2	1.89e2	5.75	1.55e1	7.27
DE/ADATE1															
Best	1.01	6.05e2	3.39e2	1.50e−2	5.24e−1	9.27	2.23e−2	3.29e2	1.01e2	1.30e2	1.00e2	1.00e2	3.69e2	4.52e2	4.90e2
Worst	2.76	8.79e4	1.50e4	1.20	1.37e4	1.98e4	1.22e2	3.29e2	1.25e2	2.01e2	1.00e2	4.01e2	4.93e2	9.99e2	1.05e3
Median	1.83	6.13e3	5.72e3	1.37e−1	1.04e3	1.31e3	4.50e−1	3.29e2	1.15e2	1.91e2	1.00e2	3.09e2	3.78e2	6.62e2	5.40e2
Mean	1.86	1.43e4	6.17e3	3.10e−1	2.35e3	2.91e3	2.81	3.29e2	1.15e2	1.86e2	1.00e2	2.99e2	3.82e2	6.62e2	5.61e2
STD	3.47e−1	2.00e4	3.68e3	3.43e−1	2.85e3	4.07e3	1.70e1	4.13e−9	4.28	1.83e1	2.78e−2	4.65e1	2.52e1	1.16e2	8.29e1
DE/ADATE2															
Best	7.40e−1	4.25e2	1.31e1	2.94e−2	2.66	4.38	1.03e−2	3.29e2	1.02e2	1.26e2	1.00e2	1.43e2	3.69e2	3.48e2	5.09e2
Worst	2.66	8.10e4	1.63e4	1.45	8.23e3	1.84e4	1.19e2	3.29e2	1.22e2	2.02e2	1.00e2	3.34e2	5.36e2	1.34e3	1.15e3
Median	1.77	4.97e3	5.73e3	1.43e−1	7.43e2	1.48e3	4.40e−1	3.29e2	1.13e2	1.98e2	1.00e2	3.09e2	3.78e2	5.71e2	6.02e2
Mean	1.73	8.56e3	6.70e3	3.37e−1	1.61e3	3.24e3	5.01	3.29e2	1.13e2	1.84e2	1.00e2	3.04e2	3.94e2	5.88e2	6.47e2
STD	4.99e−1	1.22e4	4.19e3	3.89e−1	2.20e3	4.02e3	1.72e1	5.74e−14	5.33	2.23e1	2.22e−2	3.12e1	4.15e1	1.78e2	1.45e2

Table 5.7: Performance of mutation operators on CEC 2014 benchmark functions 1 to 15 with 30 dimensions using the parameters performing best across all CEC 2014 functions as measured by the sum of all medians.

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
DE/Rand/1															
Best	6.76e3	0.00	0.00	9.21e−2	2.04e1	0.00	0.00	1.37e1	1.55e2	7.16e2	5.73e3	1.16	2.36e−1	2.19e−1	1.34e1
Worst	2.71e5	2.84e−14	5.68e−14	6.34e1	2.10e1	1.12e1	9.86e−3	1.64e2	1.98e2	4.31e3	7.20e3	3.04	4.45e−1	6.98e−1	1.68e1
Median	6.19e4	0.00	0.00	5.38e−1	2.09e1	3.46	0.00	7.10e1	1.78e2	2.11e3	6.74e3	2.41	3.22e−1	2.88e−1	1.50e1
Mean	7.93e4	1.11e−15	1.00e−14	7.92	2.09e1	3.76	3.38e−4	7.08e1	1.77e2	2.19e3	6.64e3	2.38	3.32e−1	3.09e−1	1.51e1
STD	5.51e4	5.57e−15	2.19e−14	2.05e1	8.96e−2	2.69	1.71e−3	3.05e1	1.02e1	8.56e2	3.82e2	3.17e−1	4.68e−2	9.48e−2	8.93e−1
DE/Best/1															
Best	3.36e3	0.00	0.00	3.49e−2	2.08e1	1.80e−4	0.00	8.66e1	1.56e2	2.19e3	6.23e3	1.60	2.65e−1	2.03e−1	1.33e1
Worst	2.22e5	2.84e−14	5.68e−14	6.34e1	2.10e1	1.60e1	1.14e−13	1.79e2	1.99e2	5.36e3	7.36e3	3.05	4.84e−1	3.39e−1	1.85e1
Median	5.10e4	0.00	0.00	3.55e−1	2.09e1	5.06	0.00	1.39e2	1.81e2	4.29e3	6.76e3	2.31	3.72e−1	2.81e−1	1.58e1
Mean	6.47e4	6.13e−15	3.34e−15	2.84	2.09e1	5.36	2.23e−15	1.38e2	1.79e2	4.23e3	6.77e3	2.35	3.77e−1	2.80e−1	1.57e1
STD	5.28e4	1.18e−14	1.35e−14	1.24e1	5.46e−2	3.24	1.59e−14	1.95e1	9.89	6.96e2	2.53e2	3.19e−1	4.61e−2	3.22e−2	8.86e−1
DE/current-to-best/1															
Best	1.72e2	2.84e−14	2.54e−11	3.65e−5	2.06e1	8.38	2.27e−13	2.89e1	3.38e1	9.39e2	1.89e3	1.57	2.97e−1	2.12e−1	4.40
Worst	1.35e6	2.72e−10	9.75e−2	1.62e2	2.11e1	2.35e1	2.48e−1	1.25e2	1.74e2	5.81e3	7.30e3	2.93	9.49e−1	1.15	3.94e2
Median	1.50e4	1.42e−13	3.26e−6	4.08	2.09e1	1.61e1	1.97e−2	6.50e1	8.56e1	2.14e3	3.58e3	2.46	4.72e−1	3.37e−1	1.17e1
Mean	1.26e5	1.33e−11	2.28e−3	2.78e1	2.09e1	1.63e1	3.35e−2	6.86e1	9.30e1	2.38e3	4.42e3	2.41	5.00e−1	5.27e−1	2.20e1
STD	2.94e5	4.31e−11	1.37e−2	3.61e1	7.73e−2	3.19	4.22e−2	2.06e1	3.16e1	9.85e2	1.96e3	3.23e−1	1.39e−1	2.83e−1	5.40e1
DE/rand/2															
Best	1.07e4	2.84e−14	4.06e−4	2.23e−6	2.04e1	2.09	1.14e−13	1.30e2	1.44e2	3.35e3	1.38e3	1.58	2.30e−1	2.32e−1	1.23e1
Worst	2.38e5	1.62e−12	2.39e2	7.77e1	2.10e1	2.27e1	4.18e−2	1.87e2	2.04e2	6.31e3	7.37e3	2.96	4.94e−1	7.24e−1	1.94e1
Median	3.24e4	2.84e−14	2.61	3.82e−1	2.09e1	5.98	3.41e−13	1.57e2	1.85e2	4.87e3	6.81e3	2.47	3.48e−1	3.02e−1	1.53e1
Mean	5.17e4	1.02e−13	2.10e1	5.92	2.09e1	6.98	5.84e−3	1.57e2	1.82e2	4.76e3	6.50e3	2.45	3.51e−1	3.20e−1	1.54e1
STD	4.81e4	2.35e−13	4.63e1	1.85e1	1.11e−1	4.51	9.20e−3	1.31e1	1.36e1	6.42e2	9.76e2	3.10e−1	5.97e−2	9.72e−2	1.34
DE/best/2															
Best	1.64e4	2.84e−14	2.44e−4	5.36e−3	2.05e1	8.24e−3	0.00	1.32e2	1.54e2	3.31e3	1.32e3	1.44	1.74e−1	2.36e−1	1.27e1
Worst	3.98e5	2.85e3	8.39e2	7.67e1	2.10e1	1.71e1	9.81e−2	1.85e2	2.00e2	5.97e3	7.29e3	3.16	4.91e−1	7.69e−1	1.77e1
Median	9.61e4	8.53e−13	1.19e1	5.93e−1	2.09e1	6.39	2.27e−13	1.59e2	1.83e2	4.80e3	6.72e3	2.50	3.22e−1	3.14e−1	1.56e1
Mean	1.35e5	7.46e1	7.62e1	1.00e1	2.09e1	7.33	6.85e−3	1.59e2	1.82e2	4.81e3	6.48e3	2.45	3.29e−1	3.47e−1	1.56e1
STD	1.05e5	4.02e2	1.51e2	2.34e1	9.82e−2	4.22	1.51e−2	1.27e1	1.07e1	6.05e2	1.01e3	3.33e−1	5.78e−2	1.23e−1	1.12
DE/Randrl															
Best	3.36e3	0.00	5.68e−14	1.34e−4	2.04e1	2.54	0.00	9.95	1.49e1	4.69e2	1.29e3	1.82e−1	1.73e−1	2.11e−1	2.02
Worst	7.16e4	5.68e−14	7.62e−12	6.50e1	2.10e1	1.40e1	3.69e−2	3.78e1	1.89e2	2.09e3	7.20e3	2.73	3.94e−1	7.60e−1	1.64e1
Median	1.77e4	2.84e−14	1.14e−13	9.39e−2	2.08e1	7.03	1.14e−13	2.29e1	2.79e1	9.52e2	3.23e3	1.47	2.82e−1	3.01e−1	1.41e1
Mean	2.22e4	2.79e−14	7.35e−13	7.66	2.08e1	6.98	3.72e−3	2.30e1	4.19e1	1.04e3	4.06e3	1.45	2.85e−1	3.48e−1	1.17e1
STD	1.74e4	1.20e−14	1.68e−12	2.07e1	1.38e−1	2.61	7.61e−3	6.52	4.33e1	3.77e2	1.89e3	8.21e−1	4.83e−2	1.38e−1	4.76
DE/ADATE1															
Best	9.33e4	2.15e1	2.64e2	3.46e−2	2.08e1	4.76e−1	1.14e−13	5.97	5.97	1.60e1	5.25e3	1.72	1.14e−1	2.15e−1	1.03e1
Worst	1.08e6	3.18e4	3.38e4	1.36e2	2.10e1	5.50	3.68e−2	2.60e1	1.54e2	3.82e2	6.98e3	3.03	3.00e−1	5.36e−1	1.56e1
Median	3.96e5	1.17e4	7.17e3	2.15	2.09e1	2.62	2.27e−13	1.09e1	1.15e2	1.37e2	6.48e3	2.42	2.15e−1	3.10e−1	1.32e1
Mean	4.30e5	1.30e4	9.32e3	2.92e1	2.09e1	2.70	3.33e−3	1.19e1	8.80e1	1.38e2	6.46e3	2.40	2.12e−1	3.09e−1	1.33e1
STD	2.29e5	9.06e3	8.49e3	4.32e1	4.79e−2	1.31	7.64e−3	4.41	5.68e1	1.08e2	3.21e2	3.21e−1	3.54e−2	5.20e−2	1.05
DE/ADATE2															
Best	7.31e4	2.20e3	5.99e2	7.78e−4	2.08e1	4.43	1.14e−13	9.95	1.39e1	3.64	5.90e2	1.71	1.59e−1	1.60e−1	2.00
Worst	7.77e5	2.47e4	2.67e4	1.32e2	2.10e1	1.17e1	4.41e−2	3.78e1	3.78e1	4.85e2	6.86e3	2.91	3.27e−1	3.32e−1	1.70e1
Median	2.34e5	1.03e4	5.75e3	2.33	2.10e1	7.94	3.41e−13	2.29e1	2.39e1	2.44e2	5.89e3	2.53	2.54e−1	2.35e−1	8.46
Mean	2.81e5	1.01e4	7.23e3	2.69e1	2.09e1	8.05	6.89e−3	2.36e1	2.50e1	2.15e2	4.73e3	2.45	2.49e−1	2.42e−1	8.73
STD	1.62e5	4.76e3	5.78e3	3.42e1	4.94e−2	1.71	1.30e−2	6.93	5.97	1.22e2	2.25e3	2.71e−1	3.58e−2	4.38e−2	4.76

Table 5.8: Performance of mutation operators on CEC 2014 benchmark functions 16 to 30 with 30 dimensions using the parameters performing best across all CEC 2014 functions as measured by the sum of all medians.

	16	**17**	**18**	**19**	**20**	**21**	**22**	**23**	**24**	**25**	**26**	**26**	**28**	**29**	**30**
DE/Rand/1															
Best	1.18e1	2.08e2	6.27	3.06	6.43	2.97	2.23e1	3.15e2	2.00e2	2.03e2	1.00e2	3.00e2	7.13e2	6.68e2	5.06e2
Worst	1.29e1	1.84e3	6.08e1	6.93	3.87e1	9.42e2	3.96e2	3.15e2	2.30e2	2.04e2	1.00e2	5.03e2	9.57e2	1.20e7	5.10e3
Median	1.26e1	1.32e3	5.14e1	4.74	1.03e1	4.52e2	4.49e1	3.15e2	2.24e2	2.03e2	1.00e2	3.37e2	8.25e2	7.28e2	1.12e3
Mean	1.25e1	1.24e3	4.88e1	4.93	1.40e1	3.95e2	8.12e1	3.15e2	2.23e2	2.03e2	1.00e2	3.50e2	8.33e2	6.26e5	1.65e3
STD	2.42e−1	4.00e2	1.12e1	7.71e−1	8.57	2.97e2	8.21e1	1.72e−13	6.10	2.07e−1	5.09e−2	5.96e1	3.97e1	2.54e6	1.10e3
DE/Best/1															
Best	1.22e1	8.32e2	3.98e1	4.11	2.75e1	2.22e2	2.51e1	3.15e2	2.00e2	2.03e2	1.00e2	3.00e2	7.73e2	3.31e2	4.49e2
Worst	1.30e1	1.90e3	6.59e1	6.72	4.46e1	9.38e2	1.99e2	3.15e2	2.30e2	2.03e2	1.00e2	5.94e2	8.95e2	8.65e6	3.96e3
Median	1.27e1	1.53e3	5.41e1	4.91	3.74e1	7.06e2	4.52e1	3.15e2	2.22e2	2.03e2	1.00e2	3.51e2	8.22e2	7.15e2	1.10e3
Mean	1.27e1	1.51e3	5.43e1	4.96	3.69e1	6.85e2	6.47e1	3.15e2	2.17e2	2.03e2	1.00e2	3.47e2	8.24e2	1.70e5	1.38e3
STD	1.69e−1	1.85e2	5.22	5.30e−1	4.20	1.70e2	4.99e1	1.72e−13	1.02e1	9.32e−2	4.31e−2	5.51e1	2.28e1	1.21e6	8.17e2
DE/current-to-best/1															
Best	1.10e1	8.82e2	6.16e1	5.13	4.00e1	5.43e2	2.21e1	3.15e2	2.26e2	2.03e2	1.00e2	4.01e2	9.12e2	3.97e2	1.29e3
Worst	1.25e1	1.63e4	2.52e4	6.77e1	6.17e3	3.48e4	9.49e2	3.17e2	2.61e2	2.18e2	3.63e2	1.06e3	2.37e3	2.05e7	2.63e4
Median	1.17e1	1.93e3	1.96e2	1.03e1	2.31e2	9.90e2	2.92e2	3.15e2	2.48e2	2.05e2	1.01e2	8.49e2	1.37e3	9.51e2	3.35e3
Mean	1.18e1	2.66e3	1.02e3	1.44e1	8.31e2	2.22e3	3.28e2	3.15e2	2.48e2	2.07e2	1.33e2	8.33e2	1.39e3	4.13e6	4.36e3
STD	3.64e−1	2.83e3	3.73e3	1.54e1	1.46e3	5.66e3	2.21e2	3.51e−1	6.48	3.53	6.66e1	1.61e2	3.48e2	5.96e6	4.20e3
DE/rand/2															
Best	1.16e1	9.62e2	5.80	3.53	1.20e1	2.61e2	2.44e1	3.15e2	2.22e2	2.03e2	1.00e2	3.06e2	7.38e2	3.98e2	7.90e2
Worst	1.30e1	1.50e5	5.62e2	7.87e1	5.36e3	2.75e4	5.66e2	3.15e2	2.51e2	2.08e2	3.11e2	7.29e2	1.86e3	1.62e7	8.06e4
Median	1.25e1	8.44e3	9.82e1	6.45	1.27e2	2.26e3	1.51e2	3.15e2	2.28e2	2.04e2	1.00e2	4.66e2	9.38e2	1.19e3	3.46e3
Mean	1.24e1	1.49e4	1.35e2	7.98	4.95e2	4.41e3	1.89e2	3.15e2	2.31e2	2.04e2	1.05e2	4.83e2	1.00e3	3.78e6	5.47e3
STD	3.35e−1	2.52e4	1.23e2	1.02e1	9.46e2	5.80e3	1.69e2	1.72e−13	7.27	1.10	2.94e1	1.01e2	2.07e2	5.13e6	1.18e4
DE/best/2															
Best	1.14e1	1.93e3	5.55e1	4.26	1.70e1	2.71e1	2.47e1	3.15e2	2.22e2	2.03e2	1.00e2	3.00e2	7.40e2	7.46e2	5.40e2
Worst	1.31e1	1.31e5	9.73e3	9.25	7.80e3	9.01e4	6.61e2	3.15e2	2.42e2	2.07e2	2.94e2	9.60e2	2.16e3	2.31e7	4.53e4
Median	1.26e1	8.94e3	9.25e1	5.89	2.52e2	2.12e3	7.19e1	3.15e2	2.27e2	2.03e2	1.00e2	4.40e2	9.14e2	1.28e3	2.93e3
Mean	1.25e1	1.75e4	3.23e2	6.16	7.34e2	5.81e3	1.53e2	3.15e2	2.30e2	2.04e2	1.04e2	4.64e2	1.01e3	2.54e6	4.23e3
STD	3.41e−1	2.26e4	1.35e3	1.28	1.33e3	1.29e4	1.64e2	1.72e−13	5.92	8.17e−1	2.72e1	1.04e2	2.72e2	4.83e6	6.31e3
DE/Randrl															
Best	1.11e1	1.88e2	4.48	2.23	4.55	9.41	2.23e1	3.15e2	2.23e2	2.03e2	1.00e2	3.70e2	8.10e2	4.78e2	6.28e2
Worst	1.30e1	2.74e4	2.21e2	7.01	1.57e2	7.54e2	6.38e2	3.15e2	2.43e2	2.07e2	3.22e2	7.16e2	1.45e3	1.52e7	4.57e4
Median	1.22e1	9.62e2	2.13e1	4.52	1.10e1	2.65e2	1.97e2	3.15e2	2.37e2	2.03e2	1.00e2	4.49e2	8.91e2	8.01e2	2.67e3
Mean	1.22e1	2.37e3	3.29e1	4.47	1.59e1	2.83e2	2.46e2	3.15e2	2.34e2	2.04e2	1.13e2	4.70e2	9.32e2	2.47e6	3.78e3
STD	3.70e−1	4.52e3	3.66e1	1.14	2.23e1	2.12e2	1.61e2	1.72e−13	6.66	9.10e−1	5.09e1	7.84e1	1.25e2	4.50e6	6.25e3
DE/ADATE1															
Best	9.30	1.92e4	8.18	3.18	5.43e2	9.55e3	2.24e1	3.15e2	2.24e2	2.03e2	1.00e2	3.17e2	7.74e2	1.06e3	1.80e3
Worst	1.18e1	6.33e5	2.98e3	7.05e1	1.11e4	5.50e5	4.69e2	3.15e2	2.44e2	2.13e2	2.00e2	5.17e2	9.38e2	2.93e3	5.36e3
Median	1.10e1	1.85e5	3.78e2	5.75	3.70e3	1.11e5	6.84e1	3.15e2	2.39e2	2.07e2	1.00e2	4.03e2	8.71e2	1.53e3	3.89e3
Mean	1.09e1	2.13e5	7.51e2	7.07	4.24e3	1.66e5	1.27e2	3.15e2	2.37e2	2.07e2	1.02e2	4.04e2	8.73e2	1.74e3	3.75e3
STD	5.59e−1	1.37e5	8.38e2	9.12	2.54e3	1.25e5	1.15e2	1.99e−7	5.80	2.38	1.40e1	4.51e1	3.32e1	5.94e2	8.94e2
DE/ADATE2															
Best	9.40	1.48e4	3.78e1	3.95	8.12e2	3.36e3	2.71e1	3.15e2	2.24e2	2.06e2	1.00e2	3.95e2	7.99e2	9.76e2	1.60e3
Worst	1.17e1	2.31e5	3.54e3	6.31e1	1.43e4	2.95e5	6.38e2	3.15e2	2.42e2	2.18e2	2.00e2	6.54e2	9.82e2	8.40e6	4.42e3
Median	1.04e1	1.18e5	3.01e2	6.07	3.94e3	1.08e5	1.60e2	3.15e2	2.26e2	2.11e2	1.59e2	5.27e2	8.71e2	1.42e3	3.18e3
Mean	1.04e1	1.24e5	5.71e2	7.61	4.40e3	1.26e5	2.19e2	3.15e2	2.27e2	2.11e2	1.55e2	5.30e2	8.72e2	1.66e5	3.17e3
STD	4.79e−1	5.38e4	7.37e2	8.09	2.69e3	7.33e4	1.16e2	1.72e−13	4.11	2.03	4.49e1	5.21e1	3.46e1	1.18e6	6.04e2

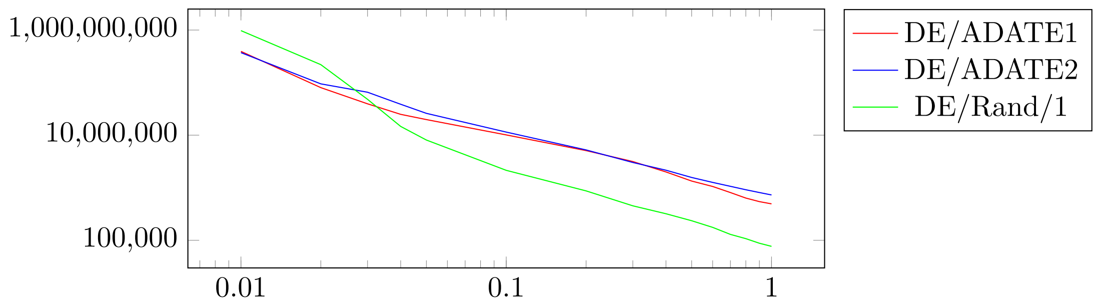


Figure 5.2: The sum of the medians of all 30-dimensional functions plotted against total function evaluations used with the best performing configuration for that number of function evaluations.

5.3 Conclusions

This chapter reported on an experiment that tried to improve the original DE algorithm using ADATE. After considering both the selection and crossover steps. The mutation operator DE/rand/1 was chosen to be improved because even small modifications to the mutation can have a significant impact on the search performance. In addition to the DE/rand/1 being both simple, and its performance having been extensively studied.

The chapter also discussed how synthesized programs should be graded, with three options explored, namely, optimizing general machine learning models, landscape generators, and synthetic optimization functions. The latter of which was chosen because of quick evaluation time and it being easier to represent most types of problems.

While the experiment did not succeed at creating an improved mutation operator as it became worse than the starting point when the number of generations was increased for testing. It shows that ADATE is capable of modifying the mutation operator to find improvements on the training and validation problems.

Although the specialization was unintentional in this case, this could be used to create specialized versions of DE to solve niche problems.

Chapter 6

Improving Competitive Differential Evolution

The goal of this chapter is to increase our understanding of how synthesized modifications to DE should be evaluated in ADATE by trying to improve the pool of competing strategies in the CDE variation of DE. The variant of CDE used in the experiment is based on the recent b3e3pbest pool of mutation strategies[20].

The experiment improves on the methodology developed to improve the standard DE algorithm in the previous chapter but significantly enhances the training and validation sets by adding more base functions, adding rotation and permutation of components, and increasing the number of generations for each problem.

This chapter is organized as follows: Section 6.1 describes the CDE algorithm and the b3e3pbest pool of mutation strategies, Section 6.2 describes the experiment, Section 6.3 reports the results of the experiment, and Section 6.4 concludes this chapter.

6.1 Competitive differential evolution

CDE is an adaptive variation of DE originally published by Tvrdk [49] in 2006 based on the competitive mechanics introduced in 2002[83]. The search adapt to the fitness landscape by using a set of H competing strategies to generate new trial vectors. A strategy is randomly selected for each individual in each generation, with more successful strategies being more likely to be chosen.

A strategy h is selected with a probability q_h calculated as:

$$q_h = \frac{n_h}{\sum_{j=1}^{H} n_j}. \tag{6.1}$$

Here, n_j is the number of times strategy h has produced trial vectors that have been accepted into the population. n_h is initialized to a constant value, usually equal to 2, which can be adjusted to control how quickly q_h changes in response to successful uses of strategies at the beginning of the search or after a reset. To prevent strategies from being dominated by others, all n_h are reset to their initial value whenever q_h decreases below a limit $\delta = \frac{1}{5h}$ for any h.

Multiple improved pools of strategies have been suggested [20], [84]–[87]. Variations have also been used to solve constraint satisfiability problems [88]. The pool used in this article is b3e3pbest, which consists of the strategies listed in Table 6.1.

Table 6.1: Pool of strategies in b3e3pbest with values for problems of 30-dimensions[20].

Mutation operator	**Crossover operator**	**F**	**CR**
DE/current-to-pbest/1	None	0.5	NA
DE/randrl/1	Binomial crossover	0.8	0.0
DE/randrl/1	Binomial crossover	0.8	0.5
DE/randrl/1	Binomial crossover	0.8	1.0
DE/randrl/1	Exponential crossover	0.8	0.8815
DE/randrl/1	Exponential crossover	0.8	0.9488
DE/randrl/1	Exponential crossover	0.8	0.9801

6.1.1 Current-to-pbest/1

The DE/current-to-pbest/1 improves on the DE/current-to-best/1 mutation to reduce the problem of getting stuck in local optima as discussed in Section 3.2.3. The solution is to select the guiding individual probabilistically from the $100p\%$ best individuals in the population:

$$\mathbf{v_i^{(t)}} = \mathbf{x_i^{(t)}} + F\left(\mathbf{x_{pbest}^{(t)}} - \mathbf{x_i^{(t)}}\right) + F\left(\mathbf{x_{R_1}^{(t)}} - \mathbf{x_{R_2}^{(t)}}\right) \tag{6.2}$$

where $\mathbf{x_{pbest}^{(t)}}$ is selected from the $100p\%$ best individuals. $\mathbf{x_{R_1}^{(t)}}$ is selected form the population P such that $P(R_1 \neq i)$, and $\mathbf{x_{R_2}^{(t)}}$ selected from the union of the population and the archive A: $P \cup A(R_2 \neq R_1 \neq i)$.

The parameter p controls how quickly the search should converge to the optimal solution. A large p leads to the search exploring more the of search space, while a small p causes the population to converge quickly. In this experiment $p = 0.05$.

6.1.2 Archive

The external archive maintains the diversity in the population by storing individuals that were replaced by the pairwise competition between the trial vector and the population individual.

When an individual is discarded from the population, it is inserted into the archive. The archive is maintained at a maximum size of $N_A^{(t)} = a \cdot N_P^{(t)}$, a being the archive rate, by randomly dropping individuals when the size is exceeded.

6.2 Experiment

The experimental methodology used to improve CDE was based on the experiment discussed in Chapter 5, enhanced to make overfitting less likely.

Unlike the previous experiment, in which the values were normalized prior to calling the f-function and the result denormalized. This experiment has for performance reasons, changed the algorithm to optimize based on only the range $[-0.5, 0.5]$ with the values denormalized when calculating the fitness using ϕ^{-1}, the inverse of Equation (5.1).

6.2.1 The f-function

The f-function was designed to make ADATE able to find an optimal pool of mutation strategies. The starting individual is the pool b3e3pbest [20]. For brevity, some of the less relevant code in the following listing has been replaced with comments.

Listing 6.1: Function to be improved by ADATE, containing heuristics for building mutating an individual in CDE.

```
fun f( Argument : fArgument ) : fResult =
  case Argument of
    totalStrategies => strategiesResult( nseven )
  | buildDonorVector(
     Num, Rdim, Current , Xpbest , Var ,
     Col , Colfitness , Jadexr1 , Jadexr2 , Z1, Z2 ) => (
      Unpack the fitness values F1, F2, and F3 from Colfitness =>
      Unpack the first column values X1, X2, X3 from Col =>
  let
    fun randlr( Scaling ) =
      case F1 < F2 of
        true =>
        ( case F1 < F3 of
            true => X1 + Scaling*( X2-X3 )
          | false => X3 + Scaling*( X1-X2 ) )
      | false =>
        ( case F2 < F3 of
            true => X2 + Scaling*( X1-X3 )
          | false => X3 + Scaling*( X1-X2 ) )
  in
    donorVectorResult(
      case Num of
        nzero => ( Current + 0.5 * ( Xpbest - Current ) +
          0.5 * ( Jadexr1 - Jadexr2  ), noCrossover )
      | none => ( randlr 0.8, binaryCrossover 0.0 )
      | ntwo => ( randlr 0.8, binaryCrossover 0.5 )
      | nthree => ( randlr 0.8, binaryCrossover 1.0 )
      | nfour => ( randlr 0.8, exponentialCrossover 0.8815 )
      | nfive => ( randlr 0.8, exponentialCrossover 0.9488 )
      | nsix => ( randlr 0.8, exponentialCrossover 0.9801 )
      | nseven => raise NA8 )
end )
```

The pool representation allows ADATE to change the number of strategies by altering the number returned when f is called with totalStrategies. The maximum is seven strategies as limited by the number of numbers defined in the numbers data type:

```
datatype numbers = nzero | none | ntwo | nthree | nfour | nfive |
nsix | nseven
```

A strategy in the pool consists of two parts, namely, a method for calculating the new donor value and a type of crossover. The donor value is returned in the first value in donorVectorResult, the type of crossover and CR value in the second. The parameters of the buildDonorVector argument are:

Num: The id of a strategy selected probabilistically with a probability calculated by Equation (6.1).

Rdim: The number of dimensions of the problem. Included so ADATE can find general improvements involving the dimensionality of the problem. The value has been converted to a real number for simpler use in calculations.

Current: The component value of the current individual, defined as x_{ij} in the rest of the thesis.

Xpbest: The component value of a randomly chosen individual from the $100p\%$ best individuals.

Var: The standard deviation of the component, calculated based on the entire population.

Col: The component values of the population represented as a real_list. The individuals are shuffled to replicate the random individual selection used for the mutation operators.

Colfitness The fitness values of the population represented as a real_list. The values are ordered in the same order as Col.

Jadexr1: The component value of a random individual chosen from the population, but not equal to Current.

Jadexr2: The component value of a random individual chosen from the population or the archive, but not equal to either the current individual or the individual which was selected for Jadexr1.

Z1: A random number between 0 and 1 chosen from a uniform distribution.

Z2: A random number sampled from a normal distribution with $\mu = 0.5$ and $\sigma = 1$.

The crossover types available can be one of noCrossover, binaryCrossover or exponentialCrossover . The latter two taking the CR value as an argument.

The algorithm has been tested against the results published by Bujok, Tvrdk, and Polakova [20] to ensure similar performance.

6.2.2 Grading synthesized programs

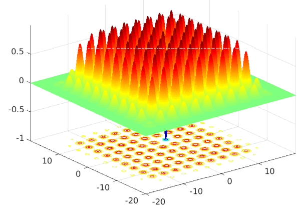
The number of generations and problems have been increased from the previous chapter to prevent the problem of overfitting from reoccurring. To further reduce the risk, all problems have also been randomly rotated, and the components permuted.

Creating representative problem sets

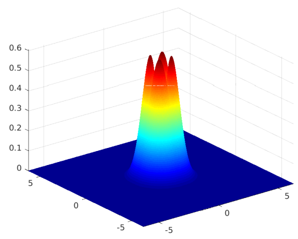
Most synthetic benchmark functions are created to study how optimization algorithms behave in specific situations. This has led to most of the functions having very different characteristics, with the majority of these being simple unimodal functions. All of the remaining interesting problems that were discovered as part of this thesis work, in addition to a few simple ones, are described in Appendix A.

The low quantity of multi-modal problems makes creating representative sets of functions difficult. To solve this, the sets of functions, excluding the functions used in CEC 2014, were recursively divided into groups: First into two groups based on whether the functions were multimodal. The multimodal functions were then subdivided based on whether they were designed for narrow niche cases. One example is the group formed by the problems f_{63}, f_{64}, and f_{66}, shown in Figure 6.1, all of which have the global optimum located inside a small, difficult to find region. The remaining problems were then tested using all of the mutation operators defined in Section 3.2 and grouped based on the relative performance of each operator.

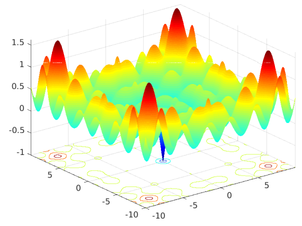
The division left some groups underrepresented, causing an unbalanced problem set. This was solved by including some of the functions initially excluded because they were used as base functions in CEC 2014. The problems used by ADATE and the testing afterward are not identical due to the different rotation and shift applied to the functions. Nevertheless, there could still be some characteristics that the improvements could exploit



(a) Xin-She Yang's Function No.01



(b) Xin-She Yang's Function No.02, this is optimized after conversion to minimization.



(c) Xin-She Yang's Function No.06

Figure 6.1: Some functions forming one niche group when creating a representative problem set.

unfairly. However, any such overfitting is easily detectable as improved performance on those functions, at the cost of degraded performance on others.

Training

The five problems used for training, all based on the same functions as in Chapter 5 are listed in Table 6.2, all evaluated using a population size of 50 individuals. The number of generations for the Sphere problem was left low because the shape of the function remains constant regardless of how far the search has progressed.

Table 6.2: Problems used for the training fitness function when improving competitive DE in ADATE, the definition of each function is available in Appendix A.

f	**Name**	D	t_{max}	$x_i \in$	**Shifted**	**Rotated**
f_1	Ackley's Function	20	500	$[-32, 32]$	Yes	Yes
f_{41}	Rastrigin Function	20	200	$[-32, 32]$	Yes	Yes
f_{53}	Sphere Function	20	50	$[-32, 32]$	Yes	Yes
f_{22}	Griewank Function	20	350	$[-32, 32]$	Yes	Yes
f_{46}	Schwefel Function	20	350	$[-500, 500]$	Yes	Yes

Validation

The entire list of problems is listed in Tables 6.3, 6.5 and 6.7.

Table 6.3: Problems used for the validation fitness when improving competitive DE in ADATE, the definition of each function is available in Appendix A.

f	**Name**	D	N_p	$x_i \in$	**Shifted**	**Rotated**
f_1	Ackley's Function	40	100	$[-32, 32]$	Yes	Yes
f_{41}	Rastrigin Function	40	80	$[-32, 32]$	Yes	Yes
f_{53}	Sphere Function	40	40	$[-32, 32]$	Yes	Yes
f_{22}	Griewank Function	40	60	$[-32, 32]$	Yes	Yes
f_{46}	Schwefel Function	40	75	$[-500, 500]$	Yes	Yes
f_{37}	Powell sum	40	40	$[-1, 1]$	Yes	Yes
f_{39}	Qing's Function	40	40	$[-100, 100]$	Yes	Yes
f_{54}	Step Function No.02	40	40	$[-100, 100]$	Yes	Yes
f_{64}	Xin-She Yang's Function No.02	40	100	$[-10, 10]$	Yes	No
f_6	Bent Cigar	40	30	$[-100, 100]$	Yes	Yes
f_{61}	Weierstrass	40	80	$[-100, 100]$	Yes	Yes
f_{33}	Modified Schwefel function	40	30	$[-100, 100]$	Yes	Yes
f_{62}	Whitley	40	40	$[-10.24, 10.24]$	Yes	Yes
f_{47}	Schwefel F2.21	40	50	$[-500, 500]$	Yes	Yes
f_{48}	Schwefel F2.26	40	70	$[-500, 500]$	Yes	Yes
f_3	Alpine Function No.01	40	40	$[-10, 10]$	Yes	Yes
f_4	Alpine Function No.02	40	40	$[0, 10]$	No	No
f_9	Brown Function	40	40	$[-1, 4]$	Yes	Yes
f_{11}	Cosine Mixture	40	100	$[-1, 1]$	Yes	Yes
f_{34}	Pathological Function	40	40	$[-100, 100]$	Yes	No
f_{40}	Quintic Function	40	40	$[-10, 10]$	Yes	Yes
f_{43}	Salomon's Function	40	40	$[-40, 40]$	Yes	Yes
f_{55}	Styblinski-Tang	40	40	$[-5, 5]$	Yes	Yes
f_{60}	W / Wavy Function	40	40	$[-\pi, \pi]$	Yes	Yes
f_{49}	Shubert Function	40	40	$[-10, 10]$	Yes	No
f_{50}	Shubert 3 Function	40	40	$[-10, 10]$	Yes	No
f_{51}	Shubert 4 Function	40	40	$[-10, 10]$	Yes	No
f_{15}	Deflected Corrugated Spring Function	40	50	$[-0, 10]$	Yes	Yes
f_{20}	Egg-Holder Function	40	50	$[-512, 512]$	Yes	Yes
f_{29}	L (or F2) Function	40	50	$[-0, 1]$	No	No
f_{30}	Lunacek's bi-Rastrigin Function	40	40	$[-32, 32]$	Yes	Yes
f_{63}	Xin-She Yang's Function No.01	40	60	$[-20, 20]$	Yes	No
f_{65}	Xin-She Yang's Function No.03	40	30	$[-2\pi, 2\pi]$	Yes	Yes
f_{66}	Xin-She Yang's Function No.06	40	100	$[-10, 10]$	Yes	No
f_{59}	Venter and Sobiezcczanski-Sobieski's Function	40	30	$[-50, 10]$	Yes	No
f_1	Ackley's Function	30	60	$[-32, 32]$	Yes	Yes
f_{41}	Rastrigin Function	30	50	$[-32, 32]$	Yes	Yes
f_{53}	Sphere Function	30	20	$[-32, 32]$	Yes	Yes
f_{22}	Griewank Function	30	70	$[-32, 32]$	Yes	Yes
f_{46}	Schwefel Function	30	75	$[-500, 500]$	Yes	Yes
f_{37}	Powell sum	30	30	$[-1, 1]$	Yes	Yes

Table 6.5: Problems used for the fitness function when improving competitive DE in ADATE, the definition of each function is available in Appendix A.

f	**Name**	D	N_p	$x_i \in$	**Shifted**	**Rotated**
f_{39}	Qing's Function	30	20	$[-100, 100]$	Yes	Yes
f_{54}	Step Function No.02	30	20	$[-100, 100]$	Yes	Yes
f_{64}	Xin-She Yang's Function No.02	30	70	$[-10, 10]$	Yes	No
f_6	Bent Cigar	30	20	$[-100, 100]$	Yes	Yes
f_{61}	Weierstrass	30	60	$[-100, 100]$	Yes	Yes
f_{33}	Modified Schwefel function	30	30	$[-100, 100]$	Yes	Yes
f_{62}	Whitley	30	30	$[-10.24, 10.24]$	Yes	Yes
f_{47}	Schwefel F2.21	30	20	$[-500, 500]$	Yes	Yes
f_{48}	Schwefel F2.26	30	30	$[-500, 500]$	Yes	Yes
f_3	Alpine Function No.01	30	50	$[-10, 10]$	Yes	Yes
f_4	Alpine Function No.02	30	30	$[0, 10]$	No	No
f_9	Brown Function	30	20	$[-1, 4]$	Yes	Yes
f_{11}	Cosine Mixture	30	30	$[-1, 1]$	Yes	Yes
f_{34}	Pathological Function	30	50	$[-100, 100]$	Yes	No
f_{40}	Quintic Function	30	30	$[-10, 10]$	Yes	Yes
f_{43}	Salomon's Function	30	30	$[-40, 40]$	Yes	Yes
f_{55}	Styblinski-Tang	30	30	$[-5, 5]$	Yes	Yes
f_{60}	W / Wavy Function	30	30	$[-\pi, \pi]$	Yes	Yes
f_{49}	Shubert Function	30	40	$[-10, 10]$	Yes	No
f_{50}	Shubert 3 Function	30	50	$[-10, 10]$	Yes	No
f_{51}	Shubert 4 Function	30	50	$[-10, 10]$	Yes	No
f_{15}	Deflected Corrugated Spring Function	30	30	$[-0, 10]$	Yes	Yes
f_{20}	Egg-Holder Function	30	40	$[-512, 512]$	Yes	Yes
f_{29}	L (or F2) Function	30	30	$[-0, 1]$	No	No
f_{30}	Lunacek's bi-Rastrigin Function	30	30	$[-32, 32]$	Yes	Yes
f_{63}	Xin-She Yang's Function No.01	30	60	$[-20, 20]$	Yes	No
f_{65}	Xin-She Yang's Function No.03	30	40	$[-2\pi, 2\pi]$	Yes	Yes
f_{66}	Xin-She Yang's Function No.06	30	40	$[-10, 10]$	Yes	No
f_{59}	Venter and Sobiezcczanski-Sobieski's Function	30	40	$[-50, 10]$	Yes	No
f_8	Brad	3	25	$-0.25 \leq x_1 \leq 0.25$ $0.01 \leq x_2, x_3 \leq 2.5$	No	No
f_{35}	Paviani	10	40	$[-2.001, 9.999]$	No	No
f_5	ANNs XOR Function	9	25	$[-1, 1]$	No	No
f_{69}	Stacked Bird, Egg-crate, Leon, Sawtoothxy	8	30	$-2\pi \leq x_1, x_2 \leq 2\pi$ $-5 \leq x_3, x_4 \leq 5$ $-1.2 \leq x_5, x_6 \leq 1.2$ $-20 \leq x_7, x_8 \leq 20$	No	No

Table 6.7: Problems used for the fitness function when improving competitive DE in ADATE, the definition of each function is available in Appendix A.

f	**Name**	D	N_p	$x_i \in$	**Shifted**	**Rotated**
f_{70}	Stacked Adjiman, Cross in Tray, Crowned Cross, Schaffer F6	8	30	$-5 \leq x_1, x_2 \leq 5$ $-15 \leq x_3, x_4 \leq 15$ $-10 \leq x_5, x_6 \leq 10$ $-100 \leq x_7, x_8 \leq 100$	No	No
f_{71}	Stacked Davis, Downhill step, Drop-wave, Siz-Hump Camel-back	8	30	$-100 \leq x_1, x_2 \leq 100$ $-10 \leq x_3, x_4 \leq 10$ $-5.1 \leq x_5, x_6 \leq 5.1$ $-5 \leq x_7, x_8 \leq 5$	No	No
f_{72}	Stacked Giunta, Hosaki, Mishra F3, Ursem F3	8	30	$-1 \leq x_1, x_2 \leq 1$ $-10 \leq x_3, x_4 \leq 10$ $-5 \leq x_5, x_6 \leq 5$ $-2.0 \leq x_7 \leq 2.0$ $-1.5 \leq x_8 \leq 1.5$	No	No
f_{10}	Bukin's Function No.06	2	25	$-15 \leq x_1 \leq -5$ $-3.0 \leq x_2 \leq 3.0$	Yes	Yes
f_{58}	Ursem-Waves Function	2	25	$-0.9 \leq x_1 \leq -1.2$ $-1.2 \leq x_2 \leq 1.2$	Yes	Yes
f_7	Bird Function	2	30	$[-2\pi, 2\pi]$	Yes	Yes
f_{19}	Egg Crate Function	2	20	$[-5, 5]$	Yes	Yes
f_{44}	Sawtoothxy Function	2	20	$[-20, 20]$	Yes	Yes
f_{18}	Drop-wave Function	2	20	$[-5.12, 5.12]$	Yes	Yes
f_{14}	Davis' Function	2	30	$[-100, 100]$	Yes	Yes
f_{28}	Leon's Function	2	20	$[-1.2, 1.2]$	Yes	Yes
f_{17}	Downhill Step Function	2	30	$[-10, 10]$	Yes	No
f_{45}	Schaffer's F6	2	30	$[-100, 100]$	Yes	No
f_{21}	Giunta's Function	2	20	$[-1, 1]$	Yes	Yes
f_{26}	Hosaki's Function	2	30	$[0, 5]$	Yes	Yes
f_{31}	Mishra's Function No.03	2	30	$[-10, 10]$	Yes	Yes
f_{57}	Ursem Function No.03	2	30	$-2.0 \leq x_1 \leq -2.0$ $-1.5 \leq x_2 \leq 1.5$	Yes	Yes
f_2	Adjiman's Function	2	30	$[-5, 5]$	Yes	Yes
f_{12}	Cross-In-Tray Function	2	30	$[-15, 15]$	Yes	No
f_{13}	Crowned Cross Function	2	30	$[-10, 10]$	Yes	No
f_{52}	Six-Hump Camel-Back Function	2	30	$[-5, 5]$	Yes	No
f_{52}	Six-Hump Camel-Back Function	2	30	$[-5, 5]$	No	Yes
f_{32}	Modified Schaffer's Function No.01	2	30	$[-100, 100]$	Yes	No
f_{36}	Peaks function	2	30	$[-4, 4]$	Yes	Yes
f_{38}	Price's Function No.02	2	30	$[-10, 10]$	Yes	Yes
f_{56}	Tsoulo's Function	2	30	$[-1, 1]$	Yes	Yes

Randomization

The following randomization strategies are used:

Permutation The components are randomly permuted on all problems. This is to prevent the improvements from overfitting to dependencies between adjacent components, as these are unlikely to be a common occurrence in practical problems.

Shift Like the previous experiment, a random shift $\mathbf{s}$ is used but here it is drawn from a uniform distribution with the range $[-0.5, 0.5]$.

Rotation The fitness landscapes are randomly rotated. The procedure to generate random rotation matrices are described in the rest of this section.

For problems with rotation, a random rotation matrix M is calculated using a procedure intended to be the same as used to generate the rotation matrices provided for the CEC 2014 problems[23]. However, there might be some differences because the procedure they use is poorly documented.

$$
\begin{aligned}
E_{ii} &= c^{\left(\frac{u_i - \min(\mathbf{u})}{\max(\mathbf{u}) - \min(\mathbf{u})}\right)} && (6.3)\\
P &= \text{GramScmidt}(A) && (6.4)\\
Q &= \text{GramScmidt}(B) && (6.5)\\
M &= PEQ && (6.6)
\end{aligned}
$$

First Equation (6.3) builds a diagonal matrix of size $D \times D$ with c being the condition number and $\mathbf{u}$ being a random vector where each component u_i is drawn from a uniform distribution between 0 and 1.

Then Equations (6.4) and (6.5) builds the two orthogonal matrices P and Q by applying Gram-Schmidt decomposition on A and B. A and B are two random matrices with elements drawn from a normal distribution.

This can cause the global optima to move outside the search bounds. To reduce the likelihood of this occurring, both the shift vector and rotation matrix is regenerated if any of the components in $M\mathbf{s}$ is outside the bounds.

The fitness function used by CDE is evaluated as:

$$f' = f(\min(\max(\phi^{-1}(M(\mathbf{x} - \mathbf{s})), \mathbf{x_{min}}), \mathbf{x_{max}})) \qquad (6.7)$$

with $\mathbf{x}$ being a candidate solution. For problems without rotation, M is equal to the identity, while for functions without shift $\mathbf{s}$ is equal 0.

Ensuring problem equality

The method used in Chapter 5 to ensure problem equality will not work with the set of problems used in this project, due to several of the problems having negative optima. This is solved by adding together the sum of the approximate cumulative densities evaluated at the achieved fitness values. The distribution is assumed to be normal, with the mean and standard deviation calculated based on 11 independent runs of the original algorithm prior to running ADATE.

Preventing overfitting

The short evaluation time required for ADATE make ensuring statistically accurate validation difficult. This problem was reduced by evaluating all problems twice using two independent tests. Each with three maximum number of generations:

$$t_{max} = \frac{m \cdot D}{N_p} \text{ for } m \in \{2000, 3000, 4000\} \tag{6.8}$$

The three increments of the number of function evaluations are intended promote solving the problem quickly and to reduce the total evaluation time.

The final grade for a program then becomes:

$$\text{grade} = \sum_{i} \sum_{t_{max}} \sum_{k=1}^{2} F_i(p_{i,k}(t_{max})) \tag{6.9}$$

Here, $F_i(\cdot)$ is the approximate cumulative density function for problem i, and $p_{i,k}(\cdot)$ is the fitness achieved when testing problem i at try k.

6.3 Results

The improved pool of strategies found by ADATE not only achieves better results but is also significantly smaller than the original. The new pool uses just four strategies, down from the original seven. The entire function encapsulating the improved pool is shown in Listing 6.2.

Listing 6.2: Function containing the entire strategies pool which has been improved by ADATE.

```
fun f Argument =
case Argument of
totalStrategies => strategiesResult( nfour )
| buildDonorVector( Num, Rdim, Current, Xpbest, Var,
                    Col, Colfitness, Jadexr1, Jadexr2, Z1, Z2 ) => (
  Unpack the first fitnesses F1, and F2 from Colfitness =>
  Unpack the first column values X1, X2, X3 from Col =>
let
fun g Scaling =
  case F1 < F2 of
    false =>
      X2 + Scaling * tanh( ( ( X1 - X3 ) / Scaling ) * Z1 )
  | true =>
      X3 + tanh( Scaling ) * Scaling * ( Jadexr1 - Jadexr2 )
in
  donorVectorResult(
    case Num of
      nzero => (
        Current + tanh( tanh( Var ) ) * ( Current - Current ) +
           tanh( X3 ) * tanh( tanh( Jadexr1 - Z1 ) ),
        binaryCrossover F2 )
    | none => ( g 0.8, binaryCrossover( tanh( tanh 0.0 ) ) )
    | ntwo => ( g( tanh( tanh 0.8 ) ), binaryCrossover 0.5 )
    | nthree => ( g 0.8, binaryCrossover 1.0 )
    | nfour => raise NA_42649
    | nfive => raise NA_4264A
    | nsix => raise NA_4264B
    | nseven => raise NA8
)
end
```

Due to limited available computation resources, ADATE was halted after having used only a small amount of the time required to find an optimal solution. This is clear from the code, which includes superfluous operations that could be eliminated entirely, such as:

$$tanh(tanh(Var)) * (Current - Current).$$

Terms like this would be removed if ADATE was given enough time and discovered no better programs in the meantime.

6.3.1 Analysis of strategy 1

The first strategy consists of binary crossover with $CR = F2$, $F2$ being the fitness value of a random individual not equal to $X3$. The binary crossover implementation would result in a single component being taken from the donor for any $CR \leq 0$ and all components taken from the donor for any $CR \geq 1$. The majority of problems used to grade synthesized programs have an optimal value on or around 0. On these problems, this causes a gradual reduction in the number of components taken from the donor as the value gets closer to the optima. On the remaining problems either all or none of the components gets taken.

The mutation, here simplified using algebraic rules:

$$\mathbf{v_i^{(t)}} = \mathbf{x_i^{(t)}} + \tanh\left(\mathbf{x_{R_1}^{(t)}}\right) * \tanh\left(\tanh\left(\mathbf{x_{j_2}^{(t)}} - rand(0,1)\right)\right) \tag{6.10}$$

with $\mathbf{v_i^{(t)}}$ being the current individual, $\mathbf{x_{R_1}^{(t)}}$ being a random individual from the population, and $\mathbf{x_{j_2}^{(t)}}$, a random individual from either the population or the archive, not equal to the current individual. The $rand(0,1)$ function produces a random number between 0 and 1 drawn from a uniform distribution.

This mutation is essentially a random walk. The lack of guidance by known good solutions and the step sizes not being scaled to more than a single individual, prevents this from being a good strategy. This assertion is backed up by the success rate of 0.2% as discussed in Section 6.3.3.

6.3.2 Analysis of strategies 2-4

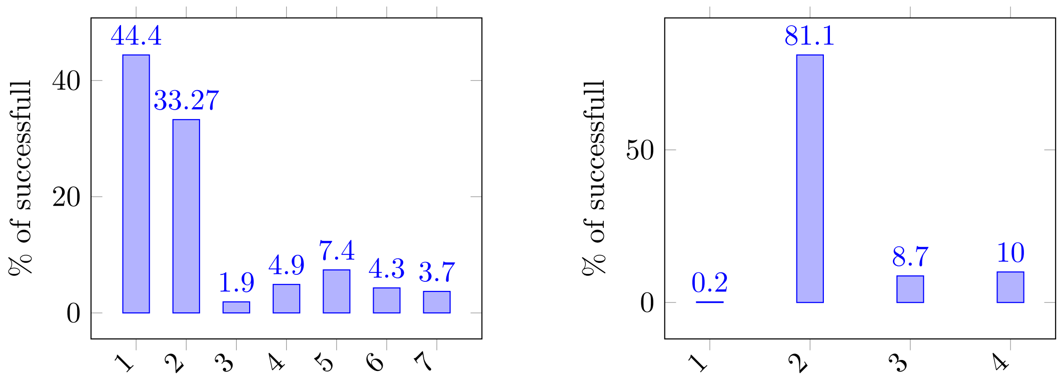
Strategies 2-4 have unchanged crossover rates, but the helper function used to build the donor have changed dramatically:

$$\mathbf{v_i^{(t)}} = \begin{cases} \mathbf{x_{R_2}^{(t)}} + F \tanh\left(\frac{\mathbf{x_{R_1}^{(t)}} - \mathbf{x_{R_3}^{(t)}}}{F} \cdot rand(0,1)\right) & \text{if } f\left(\mathbf{x_{R_1}^{(t)}}\right) < f\left(\mathbf{x_{R_2}^{(t)}}\right), \\ \mathbf{x_{R_3}^{(t)}} + (\tanh F) \cdot F \cdot \left(\mathbf{x_{j_1}^{(t)}} - \mathbf{x_{j_2}^{(t)}}\right) & \text{otherwise.} \end{cases} \tag{6.11}$$

Here $\mathbf{x_{j_1}^{(t)}}$ is a random individual from the population, not equal to the current individual. The condition $f\left(\mathbf{x_{R_1}^{(t)}}\right) < f\left(\mathbf{x_{R_2}^{(t)}}\right)$ will be true with an approximate likelihood 50% for most of the search due to $\mathbf{x_{R_1}^{(t)}}$ and $\mathbf{x_{R_2}^{(t)}}$ being randomly chosen individuals from the population.

6.3.3 Overall pool behavior

Analyzing the behavior of the original entire pool of strategies reveal a problem causing the competition within the original pool to be dysfunctional, with only two strategies dominating the search as shown in Figure 6.2a. This has become worse in the improved pool found by ADATE, with only a single strategy dominating the search, as shown in Figure 6.2b.



(a) Original

(b) Improved

Figure 6.2: Percentage of successful usages of each strategy

The problem lies in how CDE measures the successfulness of a given strategy. The success is measured by how often a strategy produces a trial vector that is successfully accepted into the population. That is $f\left(\mathbf{v}_\mathbf{i}^{(\mathbf{t})}\right) \leq f\left(\mathbf{x}_\mathbf{i}^{(\mathbf{t})}\right)$. This does not consider how much each strategy contributes to advancing the search. Fewer applications of strategies that mutate multiple components can sometimes advance the search much further than several applications which only mutate a single component. Additionally, as discussed in Section 3.3, higher crossover rates are needed for efficient navigation of non-separable problems. The high success rate of the current-to-pbest mutation which mutates all components in the original pool is explained by being good on simple landscapes.

6.3.4 Performance

The improved pool achieves better overall performance on CEC 2014 benchmark problems. As shown in Table 6.9, the improvements become larger as the number of dimensions increase. Tables 6.10 to 6.12 list the performance of the improved pool of strategies on all CEC 2014 problems. However, when compared to other algorithms in Table 7.7, the improved performance does not reach state-of-the-art levels.

Table 6.9: Comparison of the ADATE improved pool against the original b3e3pbest algorithm using the Wilcoxon Rank-sum test with $\alpha = 0.05$.

Vs. b3e3pbest		$D = 10$	$D = 30$	$D = 50$
ADATE Improvement	+ (win)	11	19	22
	- (lose)	10	3	5
	= (equal)	9	8	3

Table 6.10: Pool of strategies improved by ADATE on CEC problems with 10 dimensions using a population size of 100 individuals.

Func	Best	Worst	Median	Mean	Std
F1	5.4696E-05	2.2654E-03	3.5649E-04	4.8450E-04	4.4884E-04
F2	0.0000E+00	0.0000E+00	0.0000E+00	0.0000E+00	0.0000E+00
F3	0.0000E+00	0.0000E+00	0.0000E+00	0.0000E+00	0.0000E+00
F4	0.0000E+00	3.4780E+01	4.3354E+00	1.1847E+01	1.5751E+01
F5	9.3596E+00	2.0094E+01	2.0063E+01	1.9061E+01	2.8613E+00
F6	0.0000E+00	1.4040E-02	5.3758E-05	6.2355E-04	2.1537E-03
F7	0.0000E+00	3.3026E-02	9.5955E-03	1.1793E-02	9.8811E-03
F8	0.0000E+00	0.0000E+00	0.0000E+00	0.0000E+00	0.0000E+00
F9	2.9306E+00	7.6370E+00	5.2078E+00	5.0403E+00	1.1907E+00
F10	0.0000E+00	6.2454E-02	0.0000E+00	1.2246E-03	8.7454E-03
F11	5.2976E+01	4.5079E+02	2.2517E+02	2.3051E+02	9.7292E+01
F12	1.4912E-01	4.8205E-01	3.4187E-01	3.4109E-01	6.4149E-02
F13	1.7522E-02	1.8850E-01	1.1001E-01	1.0897E-01	3.6759E-02
F14	4.9414E-02	2.1458E-01	1.0528E-01	1.0874E-01	3.7128E-02
F15	5.1233E-01	1.1954E+00	9.0702E-01	8.8359E-01	1.4935E-01
F16	1.2421E+00	2.5521E+00	2.0773E+00	2.0599E+00	2.6026E-01
F17	8.2666E+00	5.7592E+01	2.8522E+01	2.9204E+01	1.1307E+01
F18	6.0205E-01	4.2049E+00	2.1041E+00	2.1802E+00	8.0480E-01
F19	1.3947E-01	5.7271E-01	3.2118E-01	3.3254E-01	1.0803E-01
F20	2.2045E-01	1.4308E+00	5.7551E-01	6.3176E-01	2.4470E-01
F21	4.7850E-01	7.2632E+00	1.2290E+00	1.7237E+00	1.3773E+00
F22	7.9396E-02	2.1552E+00	2.3381E-01	3.2955E-01	3.3120E-01
F23	3.2946E+02	3.2946E+02	3.2946E+02	3.2946E+02	5.7409E-14
F24	1.0660E+02	1.1709E+02	1.1225E+02	1.1226E+02	2.5002E+00
F25	1.0637E+02	2.0113E+02	1.1929E+02	1.2464E+02	2.1849E+01
F26	1.0003E+02	1.0021E+02	1.0012E+02	1.0011E+02	4.1151E-02
F27	1.0262E+00	4.0013E+02	2.4165E+00	6.2691E+01	1.4217E+02
F28	3.5683E+02	4.6960E+02	3.6885E+02	3.6905E+02	2.5302E+01
F29	2.2172E+02	2.2402E+02	2.2321E+02	2.2279E+02	7.4268E-01
F30	4.6262E+02	5.0234E+02	4.6336E+02	4.6436E+02	5.4876E+00

Table 6.11: Pool of strategies improved by ADATE on CEC problems with 30 dimensions using a population size of 100 individuals.

Func	Best	Worst	Median	Mean	Std
F1	5.0317E+02	7.2613E+04	8.5433E+03	1.4410E+04	1.4771E+04
F2	0.0000E+00	0.0000E+00	0.0000E+00	0.0000E+00	0.0000E+00
F3	0.0000E+00	0.0000E+00	0.0000E+00	0.0000E+00	0.0000E+00
F4	2.1684E-04	6.3401E+01	8.9741E-02	5.1102E+00	1.7175E+01
F5	2.0072E+01	2.0327E+01	2.0179E+01	2.0188E+01	6.4490E-02
F6	1.1291E-01	9.8681E+00	4.5187E+00	4.3890E+00	2.3607E+00
F7	0.0000E+00	0.0000E+00	0.0000E+00	0.0000E+00	0.0000E+00
F8	0.0000E+00	0.0000E+00	0.0000E+00	0.0000E+00	0.0000E+00
F9	2.2884E+01	5.1738E+01	3.5819E+01	3.7031E+01	6.8352E+00
F10	0.0000E+00	2.0819E-02	0.0000E+00	3.6740E-03	8.0157E-03
F11	9.4417E+02	2.3223E+03	1.5183E+03	1.5670E+03	2.9717E+02
F12	4.0787E-02	4.0791E-01	1.1748E-01	1.3748E-01	8.2653E-02
F13	1.1468E-01	3.9443E-01	2.8567E-01	2.8633E-01	5.4848E-02
F14	1.5117E-01	3.2053E-01	2.3124E-01	2.3373E-01	3.3940E-02
F15	1.8028E+00	5.0029E+00	3.0754E+00	3.1926E+00	7.3421E-01
F16	7.9047E+00	1.0116E+01	9.1998E+00	9.1417E+00	5.2749E-01
F17	1.4195E+02	1.1486E+03	5.1228E+02	5.6325E+02	2.2624E+02
F18	5.3315E+00	2.8751E+01	1.4937E+01	1.5526E+01	5.4755E+00
F19	2.4983E+00	5.6160E+00	3.8421E+00	3.9857E+00	7.0479E-01
F20	6.1416E+00	2.0652E+01	1.2167E+01	1.2426E+01	3.4082E+00
F21	1.9035E+01	4.9231E+02	2.2288E+02	2.2046E+02	1.3281E+02
F22	1.7649E+01	2.9045E+02	7.0048E+01	1.0269E+02	7.6626E+01
F23	3.1524E+02	3.1524E+02	3.1524E+02	3.1524E+02	1.7223E-13
F24	2.0000E+02	2.0000E+02	2.0000E+02	2.0000E+02	3.7114E-04
F25	2.0252E+02	2.0309E+02	2.0264E+02	2.0267E+02	1.2577E-01
F26	1.0011E+02	1.0043E+02	1.0026E+02	1.0026E+02	6.4711E-02
F27	3.0082E+02	4.2139E+02	4.0048E+02	3.9210E+02	2.7760E+01
F28	6.9284E+02	8.7483E+02	8.1692E+02	8.1720E+02	2.9518E+01
F29	7.1459E+02	7.1971E+02	7.1495E+02	7.1559E+02	1.4515E+00
F30	4.5879E+02	1.9827E+03	1.0465E+03	1.0107E+03	2.7610E+02

Table 6.12: Pool of strategies improved by ADATE on CEC problems with 50 dimensions using a population size of 100 individuals.

Func	Best	Worst	Median	Mean	Std
F1	1.1112E+05	6.7324E+05	3.5718E+05	3.6326E+05	1.3658E+05
F2	9.3242E-05	9.0213E+01	1.2428E-01	5.4416E+00	1.4213E+01
F3	0.0000E+00	1.2485E-08	0.0000E+00	2.4480E-10	1.7482E-09
F4	1.5047E+01	9.8397E+01	8.1331E+01	6.5334E+01	3.6063E+01
F5	2.0035E+01	2.0471E+01	2.0250E+01	2.0260E+01	1.0638E-01
F6	5.1413E+00	1.8720E+01	1.1437E+01	1.1723E+01	2.9645E+00
F7	0.0000E+00	0.0000E+00	0.0000E+00	0.0000E+00	0.0000E+00
F8	0.0000E+00	0.0000E+00	0.0000E+00	0.0000E+00	0.0000E+00
F9	6.0692E+01	1.1939E+02	8.9546E+01	9.1420E+01	1.3484E+01
F10	0.0000E+00	3.7475E-02	0.0000E+00	6.8582E-03	8.7669E-03
F11	1.8534E+03	4.5201E+03	3.5626E+03	3.5947E+03	5.1575E+02
F12	5.0120E-02	2.6643E-01	9.6178E-02	1.0405E-01	4.0576E-02
F13	2.6101E-01	5.1516E-01	3.8596E-01	3.8232E-01	4.5233E-02
F14	2.1350E-01	4.2842E-01	2.6560E-01	2.7166E-01	3.8296E-02
F15	4.2369E+00	9.5943E+00	6.2551E+00	6.5050E+00	1.1505E+00
F16	1.4721E+01	1.7862E+01	1.6937E+01	1.6823E+01	6.3577E-01
F17	1.8206E+03	5.5795E+04	7.5755E+03	9.1466E+03	8.2865E+03
F18	1.4741E+01	3.4406E+02	4.5328E+01	6.1804E+01	5.7996E+01
F19	6.8867E+00	1.3223E+01	1.0191E+01	1.0175E+01	1.2561E+00
F20	1.2698E+01	7.2005E+01	3.4216E+01	3.6165E+01	1.1400E+01
F21	3.5506E+02	3.5222E+03	9.6156E+02	1.0467E+03	5.0374E+02
F22	1.8522E+02	9.1722E+02	5.2482E+02	5.4229E+02	1.7010E+02
F23	3.4400E+02	3.4400E+02	3.4400E+02	3.4400E+02	1.1482E-13
F24	2.5300E+02	2.6713E+02	2.5629E+02	2.5802E+02	4.1979E+00
F25	2.0492E+02	2.0815E+02	2.0563E+02	2.0581E+02	6.7366E-01
F26	1.0024E+02	1.0057E+02	1.0035E+02	1.0036E+02	5.9304E-02
F27	4.0371E+02	8.2578E+02	6.2915E+02	6.3480E+02	8.9349E+01
F28	1.0705E+03	1.2782E+03	1.1859E+03	1.1833E+03	5.0225E+01
F29	7.9634E+02	1.0283E+03	8.1163E+02	8.3581E+02	5.0445E+01
F30	7.8389E+03	8.7959E+03	8.0282E+03	8.1115E+03	2.3933E+02

6.4 Conclusions

This chapter built on experiences from Chapter 5, and attempted to improve the pool of mutation strategies in CDE, using the b3e3pbest pool as an initial individual.

To prevent overfitting, the programs were evaluated using a larger set of validation problems and ensured the problem sets represented most types of function landscapes, while minimizing overlap with the test problems by recursively grouping the problems. The problem sets were also diversified by creating a different rotation and permutation for each repetition of a base problem.

Because many of the new problems have ranges that can return both negative and positive values, a new method for ensuring problem equality, by using the approximate cumulative density function of the original performance, was introduced.

The improved pool outperformed the original pool, but falls short of achieving state-of-the-art performance on the CEC 2014 set of benchmark functions. A potential reason for this could be a problem with the mechanics governing the competition among strategies. By not considering strategies that contribute considerably to the search as competitive against strategies with a low success rate but higher overall contribution is prevented from being competitive.

Chapter 7

Improving LSHADE-EpSin Differential Evolution

This chapter builds on the knowledge gained in the previous two chapters to improve the mutation operator and scale factor calculation heuristics in the state-of-the-art LSHADE-EpSin optimization algorithm[89].

The experiment further enhances the statistical certainty of the evaluations of synthesized programs by increasing the allowed number of fitness function evaluations, and the number of repetitions each program is tested on the problems. While these changes increase the total computation time required for each program, some of this is offset by reducing the dimensionality of the problems.

The chapter is structured as follows: Section 7.1 describes the LSHADE-EpSin algorithm, Section 7.2 describes the experiment, Section 7.3 discusses the results, and Section 7.4 concludes this chapter.

7.1 LSHADE-EpSin

LSHADE-EpSin is a variation of DE proposed by Awad, Ali, Suganthan, *et al.* [21] which won the CEC 2016 competition. The LSHADE-EpSin algorithm uses two sinusoidal functions to calculate the scaling factor F for the first half of the search and producing new individuals by performing an independent random walk search towards the end. Additionally, it includes adaptation mechanics introduced in [89], and Linear population size reduction from [36].

This section describes the LSHADE-EpSin algorithm as implemented in this project. There are differences between the published description of the algorithm and their implementation. Here, the description follows the implementation as submitted to CEC 2016. The overall algorithm is outlined in Algorithm 6, with new functionality described in the following subsections. The parameter values are listed in Table 7.1.

7.1.1 First half of search: Sinusoidal ensemble

In the first half of the search, as determined by the number of function evaluations, $fes < \frac{fes_{max}}{2}$, at the start of the generation, the scaling factor $F_i^{(t)}$ is calculated by randomly using one of two sinusoidal functions. The first is a decreasing, non-adaptive function

Algorithm 6 The LSHADE-EpSin algorithm.

```
$P^{(1)} \leftarrow \{\mathbf{x_i^{(1)}}, \mathbf{x_i^{(1)}}, \ldots, \mathbf{x_{N_p}^{(1)}}\}$          ▷ see Section 3.1 for details.
$t \leftarrow 1$
$fes \leftarrow N_p$
Initialize memories $M_{CR}^{(t)}$, $M_F^{(t)}$ and $M_{freq}^{(t)}$ to 0.5.          ▷ see Section 7.1.2 for details.
while Termination criteria not met do
    $S_{CR} \leftarrow \emptyset, S_F \leftarrow \emptyset, S_{freq} \leftarrow \emptyset$
    $progression \leftarrow \frac{fes}{fex_{max}}$
    for $i \leftarrow 1, N_p$ do
        $r = rand(1, H)$
        if $progression < 0.5$ then
            if $rand(0,1) < 0.5$ then
                Calculate $F_i^{(t)}$ using Equation (7.1).
            else
                $freq_i^{(t)} \leftarrow randc(M_{freq,r}^{(t)}, 0.1)$
                Calculate $F_i^{(t)}$ using Equation (7.2).
            end if
        else
            $F_i^{(t)} = randc(M_{F,r}^{(t)}, 0.1)$
        end if
        if $M_{CR,r}^{(t)} = \perp$ then
            $CR_i^{(t)} = 0$
        else
            $CR_i^{(t)} = randn(M_{CR,r}^{(t)}, 0.1)$
        end if
        $\mathbf{v_i^{(t)}} \leftarrow mutate(\mathbf{x_i^{(t)}})$          ▷ see Section 6.1.1 for details.
        $\mathbf{u_i^{(t)}} \leftarrow binaryCrossover(\mathbf{x_i^{(t)}}, \mathbf{v_i^{(t)}})$          ▷ see Section 3.3 for details.
        $fes \leftarrow fes + 1$
        if $f(\mathbf{u_i^{(t)}}) < f(\mathbf{x_i^{(t)}})$ then
            Insert $\mathbf{u_i^{(t)}}$ into $P^{(t+1)}$
        else
            Insert $\mathbf{x_i^{(t)}}$ into $P^{(t+1)}$
        end if
    end for
    Reduce population size          ▷ See Section 7.1.3 for details
    Update $M_{CR,r}$, $M_{F,r}$, and $M_{freq,r}$.          ▷ see Section 7.1.2 for details
    if $N_p < 20$ for the first time then
        Execute local search as described in Section 7.1.3
    end if
    $t \leftarrow t + 1$
end while
```

Parameter	**Value**
$freq$	0.5
H	5
$N_{p,min}$	4
$N_{p,max} = N_{p,init}$	$18 \cdot D$
N_L	10
p (In Section 6.1.1)	0.11
a	1.14

Table 7.1: Parameters used in the LSHADE-EpSin algorithm.

defined as:

$$F_i^{(t)} = \frac{1}{2}\left(\sin(2\pi \cdot freq \cdot t_1 + \pi)\frac{t_{max} - t_1}{t_{max}} + 1\right). \tag{7.1}$$

Here $freq$ is the frequency of the function.

The second function is an adaptive, increasing function defined as:

$$F_i^{(t)} = \frac{1}{2}\left(\sin(2\pi \cdot freq_i^{(t)} \cdot t_1)\frac{t_1}{t_{max}} + 1\right). \tag{7.2}$$

Here $freq_i^{(t)}$ is calculated as described in Section 7.1.2.

7.1.2 Adaptation

The adaptation uses three historical memories M_{CR}, M_{freq} and M_F, all initially set to contain H entries of 0.5. For the first half of the search, the frequency and crossover rates are randomly sampled:

$$freq_i^{(t)} = randc(M_{freq,r}^{(t)}, 0.1) \tag{7.3}$$

$$CR_i^{(t)} = \begin{cases} 0 & \text{if } M_{CR,r}^{(t)} = \perp \\ randn(M_{CR,r}^{(t)}, 0.1) & \text{otherwise} \end{cases} \tag{7.4}$$

with $r \in [1, H]$ being a randomly selected integer, and $randc(x, \gamma)$ is the Cauchy distribution with location x and γ scale. The value $\perp$ is a terminal value, which will remain in the memory slot for the rest of the search. This value forces the algorithm to only mutate one component at a time when accessing that particular memory slot.

In the last part of the search, instead of calculating the scaling factor $F_i^{(t)}$ using the sinusoidal ensemble, it is calculated using Cauchy distribution:

$$F_i^{(t)} = randc(M_{F,r}^{(t)}, 0.1) \tag{7.5}$$

For each generation, successful values of $freq_i^{(t)}$, $CR_i^{(t)}$ and $F_i^{(t)}$ are recorded in S_{freq}, S_{CR} and S_F respectively. For the first half of the search, all three values are recorded, while updating S_{freq} becomes redundant for the second half.

At the end of the generation, the contents of M_{CR} and M_F are updated as follows:

$$M_{CR,k,t+1}^{(t)} = \begin{cases} mean_{WL}(S_{CR}) & \text{if } S_{CR} \neq \emptyset \land \max(S_{CR}) \neq 0 \\ \perp & \text{if } S_{CR} \neq \emptyset \\ M_{CR,k,t} & \text{otherwise} \end{cases} \tag{7.6}$$

$$M^{(t)}_{F,k,t+1} = \begin{cases} mean_{WL}(S_F) & \text{if } S_F \neq \emptyset \\ M_{F,k,t} & \text{otherwise} \end{cases} \tag{7.7}$$

$$M_{freq,k,t+1} = \begin{cases} mean_{WL}(S_{freq}) & \text{if } S_{freq} \neq \emptyset \\ M_{freq,k,t} & \text{otherwise} \end{cases} \tag{7.8}$$

with k being an integer initially set to 1, and incremented for each generation. If $k > H$, k is to 1. The function $mean_{WL}$ is the weighted Lehmer mean defined as:

$$mean_{WL}(S) = \frac{\sum_{l=1}^{|S|} w_l \cdot S_l^2}{\sum_{l=1}^{|S|} w_l \cdot S_l} \tag{7.9}$$

$$w_l = \frac{\Delta f_l}{\sum_{j=1}^{|S|} \Delta f_j} \tag{7.10}$$

$$\Delta f_l = \left| f(\mathbf{u_l^{(t)}} - \mathbf{x_l^{(t)}}) \right| \tag{7.11}$$

7.1.3 Population size reduction

The population size is gradually reduced using the techniques introduced by Tanabe and Fukunaga [36]. At each generation, a new population size is calculated using the linear equation:

$$N_P^{(t+1)} = \text{round}\left[\left(\frac{N_{P,min} - N_{P,max}}{fes_{max}}\right) \cdot fes^{(t)} + N_{P,init}\right]. \tag{7.12}$$

Here, $N_{P,max}$ and $N_{P,min}$ are the maximum and minimum population sizes, $N_{P,init}$ is the initial population size, and fes_{max} is the maximum allowed function evaluations. When $N_P^{(t+1)} < N_P^{(t)}$, the $N_P^{(t)} - N_P^{(t+1)}$ worst individuals are dropped from the population. The archive is simultaneously resized to maintain a maximum size of $N_A = a{\cdot}N_{t+1}$ by randomly dropping surplus individuals.

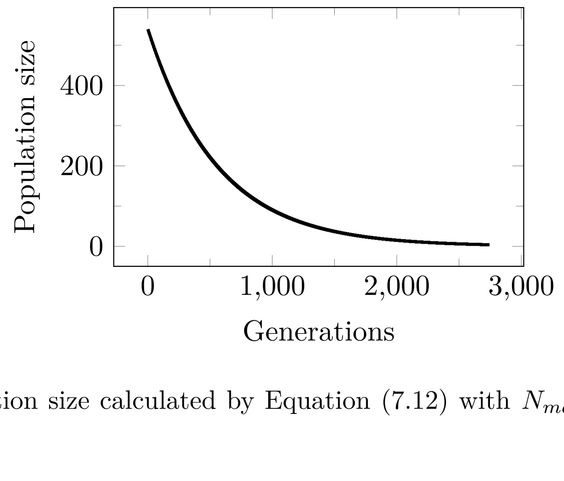


Figure 7.1: The population size calculated by Equation (7.12) with $N_{max} = 540$, $N_{min} = 4$ and $fes_{max} = 300000$.

Equation (7.12) causes the population size to be reduced following an exponential decay curve as shown in Figure 7.1.

Local search

When the population size is reduced to 20 individuals, LSHADE-EpSin uses a local search to introduce new individuals. The local search starts with a population of N_L individuals initialized randomly using the same procedure as initializing DE. For 250 generations, the following random walks are performed to produce the next generation $\mathbf{y_i}$:

$$y_{i,j} = randn(x_{best,j}, \sigma) + (randn(0,1) \cdot x_{best,j} - randn(0,1) \cdot x_{i,j}), \tag{7.13}$$

$$\sigma = \left| \frac{\log(t)}{t} (x_{i,j} - x_{best,j}) \right|. \tag{7.14}$$

Here, $x_{best,j}$ is the j'th component of the best individual in the local search population.

When the local search finishes, 10 random individuals in the population are replaced with individuals from the local search population if they are better.

7.2 Experiment

This experiment builds on the previous chapter. Therefore, only differences between the previous and this experiment are discussed to reduce redundancy.

7.2.1 The f-function

The parts of the code to be improved by ADATE were extracted into an f-function, defined to include almost all contributions of the LSHADE-EpSin algorithm. The only novelty proposed by Awad, Ali, Suganthan, *et al.* [21], that was not included is the local search algorithm.

Listing 7.1: Function to be improved by ADATE, containing heuristics mutating an individual in LSHADE-EpSin.

```
fun f( Argument : fArgument ) : fResult =
case Argument of
buildDonorVector( FValueArg, Current, Xpbest, Col,
Colfitness, Jadexr1, Jadexr2, Z1 ) => (
donorVectorResult( Current + FValueArg *
( Xpbest - Current + Jadexr1 - Jadexr2 ) ) )
| fValueArg( UsedFes, Mem1, Mem2, G, Gmax, Z3, Z4 ) => (
case realLess( UsedFes, tor( rconst( 0, 0.1, 0.5 ) ) ) of
true => (
case (
case realLess( Z3, tor( rconst( 0, 0.1, 0.5 ) ) ) of
true =>
tor( rconst( 0, 0.1, 0.5 ) ) * (
sin(
tor( rconst( 0, 0.1, 3.14159265359 ) ) * G +
tor( rconst( 0, 0.1, 3.14159265359 ) ) ) *
( ( Gmax - G ) / Gmax ) + tor( rconst( 0, 0.1, 1.0 ) ) )
| false =>
tor( rconst( 0, 0.1, 0.5 ) ) * (
sin( 6.28318530718 * Mem1 * G ) *
( G / Gmax ) + tor( rconst( 0, 0.1, 1.0 ) ) ) ) )
of FValue => fValueResult( memory( Mem1 ), memory( FValue ), FValue ) )
| false =>
fValueResult( noMemory, memory( Mem2 ), Mem2 ) )
```

The f-function consists of two parts. The current-to-pbest mutation operator, which calculate a component in the donor vector when f is called with an argument of type buildDonorVector, and the second which is triggered when f called with fValueArg, that encapsulate the logic for calculating F for the entire search.

The buildDonorVector argument consists of the following parameters:

FValueArg is a value calculated in the second part of f.

Z1 is a random number sampled from a normal distribution with $\mu = 0.5$ and $\sigma = 1$.

The remaining parameters Current, Xpbest, Jadexr1, Jadexr2 have identical descriptions as those in Section 6.2.1.

The second part is designed to give ADATE great flexibility to change the way F is calculated, and to also change, to some degree, when information is recorded into the memory. The parameters of fValueArg are:

UsedFes encapsulate the information used to switch between the sinusoidal ensemble and direct calculation of F. However, to reduce the problem of ADATE overfitting to the number of function evaluations present in the training and validation problems. UsedFes is calculated as the ratio:

$$\text{UsedFes} = \frac{fes}{fes_{max}} \tag{7.15}$$

Mem1 contains $randc(M_{mem1}, 0.1)$.

Mem2 contains $randc(M_{mem2}, 0.1)$.

G is equal to t.

Gmax is the total number of generations in the current search.

Z3 is a random number drawn from a uniform distribution between 0 and 1.

Z4 is a random number sampled from a normal distribution with $\mu = 0.5$ and $\sigma = 1$.

The return value is of type fValueResult, which allows ADATE to pass the F value as the last parameter. The first two control which information is entered into the success sets for a generation when a trial vector is accepted into the population.

Note on G and Gmax

The arguments G and Gmax are left unnormalized. This has the undesirable consequence of making it harder for ADATE to use these values because the range is far outside the range of values introduced as part of program synthesis. Additionally, the information they contain has a substantial overlap with UsedFes.

In this attempt at improving LSHADE-EpSin, the algorithm had to be reproduced exactly. While removing these arguments is desirable, several difficulties are preventing the removal without altering the algorithm:

- The halfway point in the search, as defined by the number of consumed function evaluations and by the number of generations, does not occur at the same point.

- The maximum number of generations can change depending on the start and end population sizes and the function evaluation budget. This makes it difficult to find a replacement for the frequency that is invariant in terms of these parameters.

Future work should, however, aim to remove both of these arguments at the cost of a less accurate reproduction.

7.2.2 Grading synthesized programs

The set of problems used to grade synthesized programs was altered to produce more statistically significant results. The dimensionality of many of the problems has been reduced, with the saved computation time being used to increase the number of repetitions of each problem while keeping the total evaluation time unchanged. For this experiment, additional computer resources have been made available; these resources have been used to further increase the number of seeds each problem is tested on.

The grade is the sum of all the cumulative densities when testing the problems using one or more maximum number of function evaluations:

$$\text{grade} = \sum_{i} \sum_{fes_{max}} \sum^{r} F_i(p_i(fes_{max})). \tag{7.16}$$

Here, r is the number of times each problem is repeated, $F_i(\cdot)$ is the cumulative density function for problem i, while $p_i(\cdot)$ tests the problem with an allowed budget of function evaluations fes_{max}.

Training

For training, candidate programs are evaluated using the 20 problems listed in Table 7.2 using $fes_{max} = 9000 \cdot D$ function evaluations.

Validation

The entire list of problems used for validation are similar to the once used when improving CDE which is listed in Tables 6.3, 6.5 and 6.7. The differences are:

- Problems based on f_6 are removed.
- Problems with 40 dimensions are reduced to 25 dimensions.
- Problems with 30 dimensions reduced to 15 dimensions.

The computation time saved by reducing the dimensionality of the problems are used to repeat the problems eight times for each of:

$$fes_{max} \in \{6000 \cdot D, 8000 \cdot D, 10000 \cdot D\} \tag{7.17}$$

to increase the statistical significance of the results.

Rejecting poor programs

When the f-function returns a memory(Mem) with Mem < 0.3, the program is immediately rejected. This is because LSHADE-EpSin requires the $freq$ and F values to be resampled when less than 0. Recording values less than -0.3 could therefore cause the resampling to occur prohibitively often, with the worst case being an infinite loop.

Table 7.2: Problems used for training in ADATE when improving LSHADE-EpSin.

f	**Name**	$x_i \in$	**Shifted**	**Rotated**
f_1	Ackley's Function	$[-32, 32]$	Yes	Yes
f_{41}	Rastrigin Function	$[-32, 32]$	Yes	Yes
f_{53}	Sphere Function	$[-32, 32]$	Yes	Yes
f_{22}	Griewank Function	$[-32, 32]$	Yes	Yes
f_{46}	Schwefel Function	$[-500, 500]$	Yes	Yes
f_{37}	Powell sum	$[-1, 1]$	Yes	Yes
f_{39}	Qing's Function	$[-100, 100]$	Yes	Yes
f_{54}	Step Function No.02	$[-100, 100]$	Yes	Yes
f_{64}	Xin-She Yang's Function No.02	$[-10, 10]$	Yes	No
f_{61}	Weierstrass	$[-100, 100]$	Yes	Yes
f_{33}	Modified Schwefel function	$[-100, 100]$	Yes	Yes
f_{62}	Whitley	$[-10.24, 10.24]$	Yes	Yes
f_{47}	Schwefel F2.21	$[-500, 500]$	Yes	Yes
f_{48}	Schwefel F2.26	$[-500, 500]$	Yes	Yes
f_3	Alpine Function No.01	$[-10, 10]$	Yes	Yes
f_4	Alpine Function No.02	$[0, 10]$	No	No
f_9	Brown Function	$[-1, 4]$	Yes	Yes
f_{11}	Cosine Mixture	$[-1, 1]$	Yes	Yes
f_{34}	Pathological Function	$[-100, 100]$	Yes	No
f_{40}	Quintic Function	$[-10, 10]$	Yes	Yes

7.3 Results

In the improvement found by ADATE is shown in the following listing.

Listing 7.2: The improved heuristics for mutating an individual in LSHADE-EpSin.

```
fun f Argument =
case Argument of
  buildDonorVector(
    FValueArg, Current, Xpbest, Jadexr1, Jadexr2, Z1 ) =>
   donorVectorResult(
      Current + FValueArg * ( Xpbest - Current + Jadexr1 - Jadexr2 )
| fValueArg( UsedFes, Mem1, Mem2, G, Gmax, Z3, Z4 ) =>
case
  realLess(UsedFes, tanh( tanh( 0.5 ) ) )   of
    false => fValueResult( memory( UsedFes ), memory( Mem2 ), Mem2 )
  | true =>
    fValueResult( memory( Mem1 ), memory( Z3 ),
      0.5 * ( sin( Gmax * UsedFes * UsedFes ) * ( Mem1 / G ) + 1.0 )
  )
 )
```

7.3.1 Performance

When testing the algorithm on the CEC 2014 benchmark, it is undecidable whether the algorithm is better on problems of 10 and 50 dimensions. However, the algorithm performs statistically better when tested using the Wilcoxon rank sum test with $\alpha = 0.05$ based on 51 independent trials on problems with 30 dimensions as shown in Table 7.3. The performance of the algorithm on CEC 2014 benchmark problems is shown in Tables 7.4 to 7.6. The performance is compared against several other algorithms on CEC 2014 problems with 30 dimensions in Table 7.7.

Table 7.3: Comparison of the ADATE improved LSHADE-EpSin algorithm against the original using the Wilcoxon Rank-sum test with $\alpha = 0.05$.

Vs. SHADE-EPsin		$D = 10$	$D = 30$	$D = 50$
	+ (win)	1	2	1
ADATE Improvement	- (lose)	1	0	1
	= (equal)	28	28	28

Table 7.4: Improved LSHADE-EpSin tested on CEC 2014 problems with 10 dimensions.

Func	**Best**	**Worst**	**Median**	**Mean**	**Std**
F1	0.0000E+00	0.0000E+00	0.0000E+00	0.0000E+00	0.0000E+00
F2	0.0000E+00	0.0000E+00	0.0000E+00	0.0000E+00	0.0000E+00
F3	0.0000E+00	0.0000E+00	0.0000E+00	0.0000E+00	0.0000E+00
F4	0.0000E+00	3.4780E+01	3.4780E+01	3.2734E+01	8.2650E+00
F5	3.3427E-02	2.0016E+01	1.9950E+01	1.1539E+01	9.2096E+00
F6	0.0000E+00	0.0000E+00	0.0000E+00	0.0000E+00	0.0000E+00
F7	0.0000E+00	1.7241E-02	0.0000E+00	3.3858E-04	2.4142E-03
F8	0.0000E+00	0.0000E+00	0.0000E+00	0.0000E+00	0.0000E+00
F9	1.5938E-03	2.9871E+00	1.9906E+00	1.7390E+00	6.5443E-01
F10	0.0000E+00	6.2454E-02	0.0000E+00	4.8984E-03	1.6958E-02
F11	3.5908E+00	1.2945E+02	1.7137E+01	2.7137E+01	3.1238E+01
F12	3.7850E-02	1.1647E-01	7.1112E-02	7.1771E-02	1.9624E-02
F13	1.5134E-02	7.2289E-02	4.5394E-02	4.7651E-02	1.1497E-02
F14	3.0313E-02	2.1606E-01	7.2055E-02	8.3259E-02	4.4479E-02
F15	2.5579E-01	5.9523E-01	4.0268E-01	3.9892E-01	6.5969E-02
F16	2.8986E-01	1.6706E+00	9.1551E-01	9.5962E-01	3.1770E-01
F17	0.0000E+00	1.5745E+02	1.1381E+01	2.9910E+01	4.5017E+01
F18	4.3855E-03	4.9996E-01	1.8705E-01	2.0181E-01	1.7340E-01
F19	7.0618E-04	1.0093E+00	4.9035E-02	1.7852E-01	3.1600E-01
F20	6.8061E-05	6.0600E-01	1.4414E-01	2.6193E-01	2.1770E-01
F21	2.5895E-04	1.7568E+01	3.9467E-01	1.7892E+00	4.5529E+00
F22	3.5258E-02	2.0150E+01	8.0447E-02	5.2551E-01	2.8054E+00
F23	2.0000E+02	2.0000E+02	2.0000E+02	2.0000E+02	0.0000E+00
F24	1.0000E+02	1.0941E+02	1.0705E+02	1.0633E+02	2.5740E+00
F25	1.0001E+02	2.0000E+02	1.2255E+02	1.3594E+02	3.5210E+01
F26	1.0001E+02	1.0007E+02	1.0005E+02	1.0005E+02	1.5450E-02
F27	4.1513E-01	2.0000E+02	1.1452E+00	4.0101E+01	7.9755E+01
F28	2.0000E+02	2.0000E+02	2.0000E+02	2.0000E+02	0.0000E+00
F29	1.6104E+02	2.0000E+02	2.0000E+02	1.9924E+02	5.4561E+00
F30	2.0000E+02	5.0608E+02	2.0000E+02	2.4870E+02	1.0655E+02

7.3.2 Analysis

The sinusoidal ensemble has been replaced with a single, modified sine wave that is only used in the first 43.1% of the search. Additionally, the value of Mem2 has been changed to be set to a uniform random value for the first half of the search which gets used in the last half of the search.

Written in terms of the mathematical notation used in most of the thesis, the sine wave becomes:

$$F_i^{(t)} = \frac{1}{2}\left(\sin\left(t_{max}\left(\frac{fes^{(t)}}{fes_{max}}\right)^2\right) \cdot \frac{Mem1_i^{(t)}}{t_{max}} + 1\right) \tag{7.18}$$

Recognizing that $M_{1,r}^{(t)} \in [0, 1]$, the contribution of the sine wave becomes negligible, resulting in $F_i^{(t)} \approx 0.5$.

The F5 problem is particularly interesting as neither the improved or original LSHADE-EpSin algorithm are capable of solving the problem. Looking at Figures 7.2a and 7.2b, both the improved and original algorithm produce F-values with a mean of $\bar{F} \approx 0.5$. However, the enhancement starts to produce larger F-values earlier than the original.

Table 7.5: Improved LSHADE-EpSin tested on CEC 2014 problems with 30 dimensions.

Func	Best	Worst	Median	Mean	Std
F1	0.0000E+00	0.0000E+00	0.0000E+00	0.0000E+00	0.0000E+00
F2	0.0000E+00	0.0000E+00	0.0000E+00	0.0000E+00	0.0000E+00
F3	0.0000E+00	0.0000E+00	0.0000E+00	0.0000E+00	0.0000E+00
F4	0.0000E+00	0.0000E+00	0.0000E+00	0.0000E+00	0.0000E+00
F5	2.0000E+01	2.0161E+01	2.0113E+01	2.0111E+01	2.4071E-02
F6	0.0000E+00	0.0000E+00	0.0000E+00	0.0000E+00	0.0000E+00
F7	0.0000E+00	0.0000E+00	0.0000E+00	0.0000E+00	0.0000E+00
F8	0.0000E+00	0.0000E+00	0.0000E+00	0.0000E+00	0.0000E+00
F9	7.1375E+00	1.9268E+01	1.4047E+01	1.3254E+01	2.4461E+00
F10	0.0000E+00	2.0819E-02	0.0000E+00	2.8575E-03	7.2355E-03
F11	5.1441E+02	1.5304E+03	1.1593E+03	1.1391E+03	1.9046E+02
F12	1.1662E-01	2.2506E-01	1.5904E-01	1.5959E-01	2.1099E-02
F13	1.1040E-01	1.6147E-01	1.3535E-01	1.3716E-01	1.3763E-02
F14	1.2196E-01	2.5218E-01	1.9491E-01	1.9639E-01	2.6400E-02
F15	1.8199E+00	2.8275E+00	2.3092E+00	2.3092E+00	2.3164E-01
F16	7.2095E+00	9.0607E+00	8.2675E+00	8.2198E+00	3.7834E-01
F17	2.0246E+01	4.2683E+02	1.6840E+02	1.6897E+02	9.6394E+01
F18	1.4426E+00	1.4406E+01	5.4072E+00	5.9756E+00	2.6348E+00
F19	9.0399E-01	4.5579E+00	2.7921E+00	2.8425E+00	7.7167E-01
F20	7.5101E-01	5.0629E+00	1.8125E+00	2.1794E+00	1.1657E+00
F21	1.9220E+00	2.5679E+02	4.3427E+01	8.0823E+01	7.4664E+01
F22	9.2745E+00	1.4406E+02	2.4693E+01	5.3963E+01	5.0682E+01
F23	2.0000E+02	2.0000E+02	2.0000E+02	2.0000E+02	0.0000E+00
F24	2.0000E+02	2.0000E+02	2.0000E+02	2.0000E+02	0.0000E+00
F25	2.0000E+02	2.0000E+02	2.0000E+02	2.0000E+02	0.0000E+00
F26	1.0009E+02	1.0017E+02	1.0013E+02	1.0013E+02	1.7020E-02
F27	2.0000E+02	3.0000E+02	2.0000E+02	2.0392E+02	1.9604E+01
F28	2.0000E+02	2.0000E+02	2.0000E+02	2.0000E+02	0.0000E+00
F29	2.0000E+02	2.0000E+02	2.0000E+02	2.0000E+02	0.0000E+00
F30	2.0000E+02	2.0000E+02	2.0000E+02	2.0000E+02	0.0000E+00

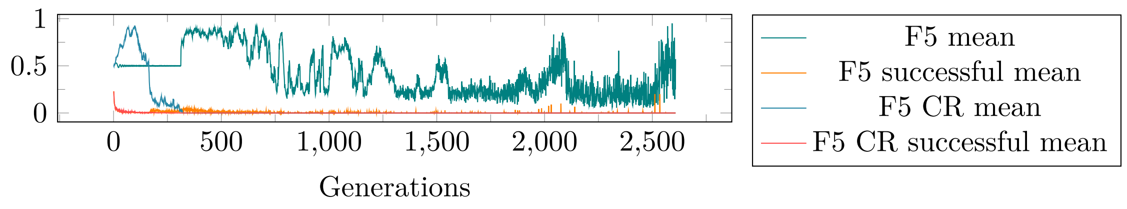
The plots also reveal a potential reason for both algorithms being unable to solve this problem. The CR value drops to 0, early in the search. LSHADE-EpSin locks the memory slot value to $\perp$ for the remainder of the search when the only CR value that produced an individual that was successfully admitted to the population, is $CR = 0$. Locking this early might limit how much the search can explore the search space.

7.4 Conclusions

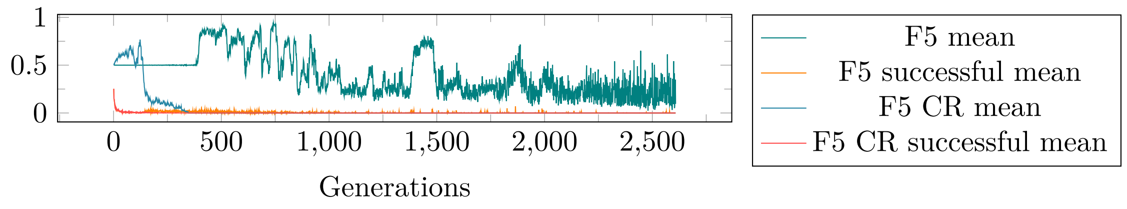
In this chapter, we improved the mutation operator and scaling factor calculation for the state of the art LSHADE-EpSin algorithm using ADATE. The improvement results in a statistically significant performance gain on problems with 30 dimensions, but there is no clear winner on problems with either 10 or 50 dimensions. In the improved algorithm, most of the sinusoidal ensemble has been removed, and the ensemble is used in a shorter period at the start of the search. Upon analyzing the improvement, it was also discovered that there was a potential problem with the base LSHADE-EpSin algorithm that could limit its performance by preventing the mutation of multiple components for most of the search.

Table 7.6: Improved LSHADE-EpSin tested on CEC 2014 problems with 50 dimensions.

Func	**Best**	**Worst**	**Median**	**Mean**	**Std**
F1	0.0000E+00	9.6799E-01	4.9655E-07	2.0916E-02	1.3546E-01
F2	0.0000E+00	0.0000E+00	0.0000E+00	0.0000E+00	0.0000E+00
F3	0.0000E+00	0.0000E+00	0.0000E+00	0.0000E+00	0.0000E+00
F4	2.0140E-01	9.8397E+01	5.9480E+00	4.8551E+01	4.8059E+01
F5	2.0170E+01	2.0310E+01	2.0260E+01	2.0257E+01	2.8293E-02
F6	7.0600E-06	3.8942E-03	3.7207E-04	6.9790E-04	8.6616E-04
F7	0.0000E+00	0.0000E+00	0.0000E+00	0.0000E+00	0.0000E+00
F8	0.0000E+00	8.4812E-08	0.0000E+00	2.1479E-09	1.2304E-08
F9	1.4184E+01	3.8396E+01	2.8759E+01	2.6720E+01	6.8397E+00
F10	3.5576E-03	1.0466E-01	4.3865E-02	5.0707E-02	2.5202E-02
F11	2.0648E+03	3.8193E+03	3.1482E+03	3.1375E+03	3.3743E+02
F12	1.6326E-01	2.6375E-01	2.0800E-01	2.0912E-01	2.5792E-02
F13	1.6483E-01	2.4341E-01	2.0247E-01	2.0386E-01	1.9419E-02
F14	1.5753E-01	2.5489E-01	1.9745E-01	2.0166E-01	2.2025E-02
F15	4.4209E+00	6.4147E+00	5.2438E+00	5.2860E+00	4.8745E-01
F16	1.5452E+01	1.7363E+01	1.6614E+01	1.6539E+01	4.5610E-01
F17	7.1674E+01	9.2081E+02	3.6121E+02	4.0135E+02	1.8502E+02
F18	7.2045E+00	3.5022E+01	1.7275E+01	1.7824E+01	6.3183E+00
F19	6.2804E+00	1.1088E+01	9.9849E+00	9.6713E+00	1.2034E+00
F20	2.4676E+00	1.2438E+01	6.1333E+00	6.5289E+00	2.1145E+00
F21	1.3623E+02	6.1989E+02	3.6324E+02	3.3213E+02	1.0375E+02
F22	2.7048E+01	2.7101E+02	6.2018E+01	1.0332E+02	7.0862E+01
F23	2.0000E+02	2.0000E+02	2.0000E+02	2.0000E+02	0.0000E+00
F24	2.0000E+02	2.0000E+02	2.0000E+02	2.0000E+02	0.0000E+00
F25	2.0000E+02	2.0000E+02	2.0000E+02	2.0000E+02	0.0000E+00
F26	1.0015E+02	1.0025E+02	1.0019E+02	1.0019E+02	1.8512E-02
F27	2.0000E+02	3.4725E+02	2.0000E+02	2.0485E+02	2.4693E+01
F28	2.0000E+02	2.0000E+02	2.0000E+02	2.0000E+02	0.0000E+00
F29	2.0000E+02	2.0000E+02	2.0000E+02	2.0000E+02	0.0000E+00
F30	2.0000E+02	2.0000E+02	2.0000E+02	2.0000E+02	0.0000E+00



(a) The mean of all and the only the successful F and CR plotted for each generation in the improved LSHADE-EpSin algorithm.



(b) The mean of all and the only the successful F and CR plotted for each generation in the original LSHADE-EpSin algorithm.

Figure 7.2: Plots of the means of F, CR, and for each generation for the improved and original algorithm.

Table 7.7: Comparison of improved LSHADE-EpSin and improved CDE against the original algorithms. The values are the mean and standard deviation when the algorithms are tested 51 times on the CEC 2014 functions with 30 dimensions.

Func	**Improved LSHADE-EpSin**	**Improved CDE**	**LSHADE-EpSin**	**b3e3pbest**
F1	**0.0000E+00** **(0.0000E+00)**	1.4410E+04 (1.4771E+04)	**0.0000E+00** **(0.0000E+00)**	9.4315E+03 (5.9990E+03)
F2	**0.0000E+00** **(0.0000E+00)**	**0.0000E+00** **(0.0000E+00)**	**0.0000E+00** **(0.0000E+00)**	**0.0000E+00** **(0.0000E+00)**
F3	**0.0000E+00** **(0.0000E+00)**	**0.0000E+00** **(0.0000E+00)**	**0.0000E+00** **(0.0000E+00)**	**0.0000E+00** **(0.0000E+00)**
F4	**0.0000E+00** **(0.0000E+00)**	5.1102E+00 (1.7175E+01)	1.5583E-08 (9.1000E-08)	1.2432E+00 (8.8779E+00)
F5	**2.0111E+01** **(2.4071E-02)**	2.0188E+01 (6.4490E-02)	**2.0119E+01** **(2.0934E-02)**	2.0320E+01 (2.8623E-02)
F6	**0.0000E+00** **(0.0000E+00)**	4.3890E+00 (2.3607E+00)	2.7688E-07 (1.9773E-06)	1.4052E+01 (2.3233E+00)
F7	**0.0000E+00** **(0.0000E+00)**	**0.0000E+00** **(0.0000E+00)**	**0.0000E+00** **(0.0000E+00)**	2.9474E-03 (4.8538E-03)
F8	**0.0000E+00** **(0.0000E+00)**	**0.0000E+00** **(0.0000E+00)**	**0.0000E+00** **(0.0000E+00)**	**0.0000E+00** **(0.0000E+00)**
F9	**1.3254E+01** **(2.4461E+00)**	3.7031E+01 (6.8352E+00)	**1.3369E+01** **(2.0754E+00)**	4.6986E+01 (7.1214E+00)
F10	**2.8575E-03** **(7.2355E-03)**	3.6740E-03 (8.0157E-03)	**2.0411E-03** **(6.2526E-03)**	4.0822E-04 (2.9153E-03)
F11	**1.1391E+03** **(1.9046E+02)**	1.5670E+03 (2.9717E+02)	**1.1829E+03** **(1.6755E+02)**	2.4720E+03 (2.4943E+02)
F12	1.5959E-01 (2.1099E-02)	**1.3748E-01** **(8.2653E-02)**	1.5979E-01 (2.2386E-02)	4.4352E-01 (5.2421E-02)
F13	**1.3716E-01** **(1.3763E-02)**	2.8633E-01 (5.4848E-02)	**1.3421E-01** **(1.8166E-02)**	2.8380E-01 (4.7173E-02)
F14	**1.9639E-01** **(2.6400E-02)**	2.3373E-01 (3.3940E-02)	**1.9783E-01** **(2.2480E-02)**	2.4253E-01 (4.1479E-02)
F15	**2.3092E+00** **(2.3164E-01)**	3.1926E+00 (7.3421E-01)	**2.2957E+00** **(3.2137E-01)**	5.6874E+00 (5.5490E-01)
F16	**8.2198E+00** **(3.7834E-01)**	9.1417E+00 (5.2749E-01)	**8.1764E+00** **(4.7404E-01)**	9.9306E+00 (3.2950E-01)
F17	**1.6897E+02** **(9.6394E+01)**	5.6325E+02 (2.2624E+02)	**1.4111E+02** **(8.4712E+01)**	1.1022E+03 (4.7227E+02)
F18	**5.9756E+00** **(2.6348E+00)**	1.5526E+01 (5.4755E+00)	**6.3116E+00** **(2.7455E+00)**	6.0158E+01 (4.3118E+01)
F19	**2.8425E+00** **(7.7167E-01)**	3.9857E+00 (7.0479E-01)	**2.7111E+00** **(6.1065E-01)**	4.9758E+00 (1.0238E+00)
F20	**2.1794E+00** **(1.1657E+00)**	1.2426E+01 (3.4082E+00)	**2.3591E+00** **(1.0403E+00)**	1.5227E+01 (5.0255E+00)
F21	**8.0823E+01** **(7.4664E+01)**	2.2046E+02 (1.3281E+02)	**8.8999E+01** **(7.2279E+01)**	2.1494E+02 (1.1635E+02)
F22	**5.3963E+01** **(5.0682E+01)**	1.0269E+02 (7.6626E+01)	**5.5032E+01** **(5.0009E+01)**	1.3378E+02 (6.0558E+01)
F23	**2.0000E+02** **(0.0000E+00)**	3.1524E+02 (1.7223E-13)	**2.0000E+02** **(0.0000E+00)**	3.1524E+02 (1.7223E-13)
F24	**2.0000E+02** **(0.0000E+00)**	2.0000E+02 (3.7114E-04)	2.0000E+02 (5.8812E-10)	2.2726E+02 (4.6729E+00)
F25	**2.0000E+02** **(0.0000E+00)**	2.0267E+02 (1.2577E-01)	**2.0000E+02** **(0.0000E+00)**	2.0405E+02 (1.1379E+00)
F26	**1.0013E+02** **(1.7020E-02)**	1.0026E+02 (6.4711E-02)	**1.0013E+02** **(1.6101E-02)**	1.0028E+02 (4.9230E-02)
F27*	2.0392E+02 (1.9604E+01)	3.9210E+02 (2.7760E+01)	**2.0000E+02** **(1.4003E-10)**	3.7227E+02 (4.9793E+01)
F28	**2.0000E+02** **(0.0000E+00)**	8.1720E+02 (2.9518E+01)	**2.0000E+02** **(0.0000E+00)**	8.2827E+02 (4.8915E+01)
F29	**2.0000E+02** **(0.0000E+00)**	7.1559E+02 (1.4515E+00)	**2.0000E+02** **(0.0000E+00)**	7.5534E+02 (3.8937E+01)
F30	**2.0000E+02** **(0.0000E+00)**	1.0107E+03 (2.7610E+02)	**2.0000E+02** **(0.0000E+00)**	1.6088E+03 (7.5520E+02)

* The result for the improved LSHADE-EpSin algorithm is due to a single bad initialization.

Chapter 8

Specializing Differential Evolution for training Spiking Neural Networks

This chapter aims to contribute to to understand how specialized versions of DE might be created for training SNNs.

The structure of the chapter is as follows: Section 8.1 provides an in-depth description of SNN by discussing neuron models, synaptic plasticity, existing supervised learning methods, and LSMs. Section 8.2 proposes a specialized DE version to train SNNs. Lastly, Section 8.3 summarizes the chapter.

8.1 Spiking neural networks

SNNs are the third generation of neural networks. The neurons in an SNN communicate using a sequence of binary events called spikes, which are produced when the membrane potential of a neuron reaches a threshold. These networks can be much more power efficient when implemented in hardware[17], and have been shown to possess the same computational power as traditional ANN using fewer gates [18]. While theoretically interesting, there are few applications in engineering contexts, apart from some limited use of LSM[19].

The lack of applications are because there currently does not exist any good training methods for SNNs, but DE algorithms could probably be successfully used to train SNNs, especially if specialized for these types of networks.

SNN are inspired by biological neural networks. A biological neuron, illustrated in figure 8.1, consists of (for our purposes) three parts. The dendrites collect incoming signals from other neurons, the soma which perform processing on these inputs and the axon serves as the output from of the neuron.

The signals neurons use to transfer information consist of a sequence of action potentials or spikes called spike trains. The neurons ensure that spikes are well separated, and it is for most neurons impossible to trigger another spike within the refractory period of the neuron. Neurons which cause an action potential is called the *presynaptic neuron*, while the receiving neuron is called the *postsynaptic neuron*.

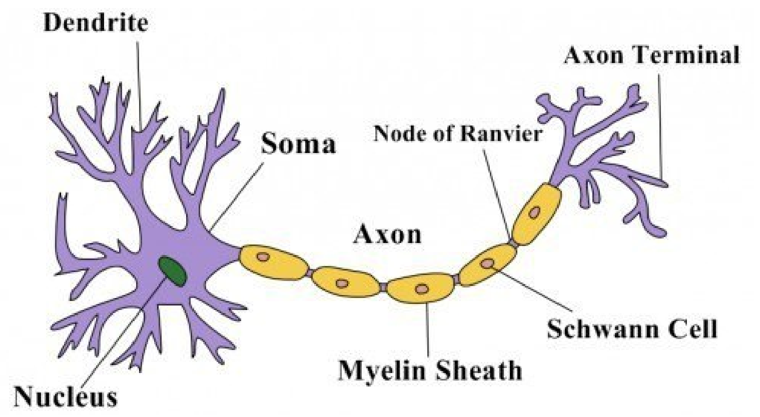


Figure 8.1: Structure of a typical neuron.

8.1.1 Neurons

There have been developed mathematical models which describe how biological neurons work which either reproduces some or all observed behaviors. The seminal paper by Izhikevich [90] describes 20 of the features observed in biological spiking neurons.

The most well-known neuron model is the Hodgkin-Huxley model which resulted from the study of a giant squid axon [91]. This model is however too complicated to be analyzed in analytically or computationally efficient enough to be used in large-scale simulations. As a result, there have been developed many simpler models which replicate one or more of the properties[90], [92]–[94]. Nevertheless, there are many additional details in biological neurons which are accounted for in the Hodgkin-Huxley model, but it is unknown whether these contribute to computation in the brain.

This section discusses some of the most popular point neuron models.

Leaky integrate and fire

The Leaky Integrate and Fire (I&F) is perhaps the most widely used spiking neuron model [95]. This simple model is attractive due to easy implementation in software with relatively efficient simulation, and because it is relatively easy to analyze the networks mathematically. The neuron can be modeled as a *leaky integrator* of the input current $I(t)$:

$$\tau_m \frac{du}{dt} = -v + RI(t), \tag{8.1}$$

where $v(t)$ represents the membrane potential at time t, τ_m is the membrane time constant and R is the resistance. The shape of spikes is not described explicitly by the model. Instead, when the membrane potential reaches the threshold v_{th}, it is reset to a lower value v_r, and the leaky integration process starts again.

While I&F is widely used, it lacks many of the biological properties that have been observed and which are modeled in more sophisticated models like the Hodgkin-Huxley model [91]. The most obvious being the lack of an absolute refractory period in which the neuron will not fire again. Additionally, it does not show adaptation, which is the observed phenomenon in which the firing frequency slows down when the neuron is exposed to a constant stimulus. Additionally, in biological neurons, the shape of the postsynaptic potential depends not only on the membrane potential but also on the internal state of the neuron. For example, a spike that arrived shortly after the neuron has fired will cause a different response due to some of the ion channels still being open.

The Izhikcvhich neuron

The Izhikcvhich neuron [90] is a model capable of exhibiting many of the features observed in biological neurons. While at the same time having a much lower computational complexity than the Hodgkin-Huxley model [91]. It is however not biologically plausible. The neuron can be described by the following two Ordinary Differential Equations (ODEs):

$$\dot{v} = 0.04v^2 + 5v + 140 - u - I, \tag{8.2}$$

$$\dot{u} = a(b \cdot v - u), \tag{8.3}$$

$$\text{if } v = 30mV \text{ then } v = c, u = u + d. \tag{8.4}$$

The model consists of two state variables, v, and u. v is the membrane potential, and u being the neuron recovery variable. The spiking condition is $v = 30mV$, the neuron fires whenever this condition is met before going back to the reset state.

The remaining variables a, b, c and d are parameters of the Izhikcvhich model. The parameter a describes the speed of recovery, smaller values equal slower recovery, and b describe how sensitive the recovery variable u is to sub-threshold fluctuations in the membrane potential v. When b is large, u and v are more tightly coupled, which can result in sub-threshold oscillations and low-threshold spiking dynamics. The parameter c is the value to which the membrane potential v is set to after a spike. Lastly, d affects the value which the recovery variable u is set to after a spike.

Resonate and fire

A resonate and fire neuron [96] is similar to an I&F with the internal state of the neuron being complex and can be described as:

$$z' = I + (b + iw)z \tag{8.5}$$

$$\text{if Im } x == a_{thresh}, \text{ then } z \leftarrow z_0(z) \tag{8.6}$$

with b, w, and a_{thresh} being parameters and $z_0(z)$, a function describing activity-dependent activity after the spike.

FitzHugh-Nagumo

The FitzHugh-Nagumo model [92] is a simplification of the Hodkin-Huxley model which models the voltage v and sodium activation, and combining the potassium activation and sodium inactivation as the recovery variable w. This can be expressed as:

$$\epsilon\dot{v} = F(v) - w + I, \tag{8.7}$$

$$\dot{w} = v - \gamma w, \tag{8.8}$$

with I being the injected current and

$$F(v) = v(1 - v)(v + a), \tag{8.9}$$

a, γ and ϵ being constants with $\epsilon \ll 1$.

Morris-Lecar

The Morris-Lecar model [93] is similar to the HodgkinHuxley but is based on the calcium instead of sodium currents provided through the two ionic conductances. The neuron can be modeled by the following differential equations:

$$C\dot{u} = I - g_1 \hat{m_0}(u)(u - V_1) - g_2 w(u - V_2) - g_L(u - V_L), \tag{8.10}$$

$$\dot{w} = -\frac{1}{\tau(u)}\left[w - w_0(u)\right]. \tag{8.11}$$

The first describe the dynamics for the membrane potential u, while the second describe the recovery variable w. The voltage dependency can be approximated by:

$$m_0(u) = \frac{1}{2}\left[1 + \tanh\left(\frac{u - u_1}{u_2}\right)\right] \tag{8.12}$$

$$w_0(u) = \frac{1}{2}\left[1 + \tanh\left(\frac{u - u_3}{u_4}\right)\right] \tag{8.13}$$

$$\tag{8.14}$$

with parameters $u_1, \ldots, u_4$. The time constant can be approximated by:

$$\tau(u) = \frac{\tau_w}{\cosh\left(\frac{u-u_3}{2u_4}\right)}, \tag{8.15}$$

with τ_w being a parameter.

Hindmarsh-Rose

The Hindmarsh-Rose model is a modification of the FitzHugh-Nagumo model which can be described by the following equations[97]:

$$\dot{x} = y - f(x) - z + I, \tag{8.16}$$

$$\dot{y} = g(x) - y, \tag{8.17}$$

$$\dot{z} = \epsilon(s(x - x_0) - z), \tag{8.18}$$

with I being the input current and:

$$f(x) = ax^3 - vx^2, \tag{8.19}$$

$$g(x) = c - dx^2. \tag{8.20}$$

The ϵ and s being constants which controls how the neuron behaves.

8.1.2 Spike response model

The most common approach to describing neuronal dynamics in an I&F network is using differential equations. However, an alternative approach is to parameterize the variables of the model as functions of time called filters. The membrane potential for neuron j can then be described by:

$$V_j(t) = \sum_{i \in \Gamma_j} \sum_{k} w_{ij}^{k} \epsilon(t - t_i^{out} - d_{ij}^{k}). \tag{8.21}$$

$\epsilon(t)$ is the post-synaptic potentials, Γ_j is the set of all pre-synaptic neurons of neuron j. w_{ij}^k is the strength of the synaptic terminal k between neurons i and j. t_i^{out} is the firing time of neuron i and d_{ij}^k is the delay of the synaptic terminal. It is assumed that:

$$\epsilon(t) = \frac{1}{\tau \exp\left(1 - \frac{1}{\tau}\right)} \tag{8.22}$$

with τ being a time constant.

8.1.3 Synaptic plasticity

A remarkable property of the mammalian brain is its ability to modify the neural circuity and thereby changing how future computation is accomplished. One of the most important mechanisms for this is synaptic plasticity which is believed to be of fundamental importance in the formation of memories in the brain. There have therefore been an enormous amount of work trying to discover the underlying mechanisms of the various types of synaptic plasticity. This section will briefly describe some of the most important concepts. A more thorough review is given in [98].

Rate based models

Rate based synaptic plasticity models assume the magnitude of changes in synaptic strength is determined by the frequency of post- and presynaptic firing measured over some time interval.

Plain Hebbian learning rule is the rule which most directly describes the fundamentally important postulate of Hebb [99] which is often paraphrased as "Cells that fire together, wire together"[100]. This can be expressed plainly in a rate based model as follows[100]:

$$\tau_w \dot{w} = v\mathbf{u} \tag{8.23}$$

with u being the presynaptic firing rate, v being the postsynaptic firing rate, τ_w being a timing constant.

In practice this rule is problematic because the connection strength is only able to increase, and never decrease. This would cause the weights to increase until an optional upper bound is reached. At which point the weights will start to saturate.

Anti-Hebbian learning is a rule first proposed by Barlow, Barlow, and Fddik [101], which can be stated similarly as Hebbian learning, but with a negative learning rate. This has the effect of making different outputs less correlated. While this is believed to be a primary principle for Purkinje cells in the cerebellum, it is unclear how general this is as Purkinje cells are inhibitory[100].

Synaptic Normalization is global constraints imposed on the synaptic strengths to prevent the postsynaptic neuron from becoming too active[100]. One common constraint is to require the sum of all synaptic weights to equal a constant value.

Oja's rule is a modification to the plain interpretation of Hebb's postulate to address the stability issues which occurs due to pure potentiating [102]. The rule is equivalent to extracting the principal component of the presynaptic input when used with a single neuron. The rule can be expressed as:

$$\Delta w_i = \alpha(x_i y - y^2 w_i) \tag{8.24}$$

with w_i being the synaptic strength from neuron i, α the learning rate, x_i is the presynaptic input and y the output of the postsynaptic neuron.

When compared to the plain interpretation of Hebb's postulate, the rule is similar, except for in including the $-y^2 w_i$ term which regulates the potentiating by reducing the potentiating when the strength or output becomes large.

Bienenstock-Cooper-Munro (BCM) rule is a synaptic update rule which uses a sliding threshold for determining the sign of the update Bienenstock, Cooper, and Munro [103].

$$\tau_w \dot{\mathbf{w}} = v\mathbf{u}(v - \theta_v) \tag{8.25}$$

with u being the presynaptic firing rate, v being the postsynaptic firing rate, τ_w being a timing constant, $\mathbf{u}$ with the threshold θ_v given by:

$$\tau_\theta \dot{\theta_v} = v^2 - \theta_v. \tag{8.26}$$

here τ_θ is a timing constant.

Izhikevich and Desai [104] has shown that a pair-based Spike-Timing-Dependent-Plasticity (STDP) which operates on the nearest spike principle, can replicate BCM behavior when the spike arrival times are distributed according to the Poisson distribution. Furthermore, Pfister and Gerstner [105] showed that triplet-based STDP could also replicate the behavior of BCM when the pre- and postsynaptic spike times follows a Poisson distribution.

Spike-Timing-Dependent-Plasticity

STDP is a partial explanation for Long-Term Potentiation (LTP) and Long-Term Depression (LTD) in biological neural networks. In STDP, if a spike arrives shortly before the neuron fires, then that particular input is strengthened. However, if the spike arrives shortly after, then that input is weakened. Thus reducing the chance of it participating in future action potentials.

Pair-based STDP The classical description of STDP is the pair-based rule which can be stated as follows:

$$\Delta w = \begin{cases} \Delta w^+ = A^+ \exp\left(\frac{-\Delta t}{\tau_+}\right) & \text{if } \Delta t > 0 \\ \Delta w^- = -A^- \exp\left(\frac{\Delta t}{\tau_-}\right) & \text{if } \Delta t \leq 0, \end{cases} \tag{8.27}$$

where $\Delta t = t_{post} - t_{pre}$ is the time difference between two pre- and postsynaptic spikes. This will result in a potentiation of the weights if a postsynaptic spike arrives in a specific time window τ_+ after the occurrence of a presynaptic spike. Similarly, a depression will occur if a presynaptic spike arrives in the time window τ_-. The amount of depression and potentiation depends on the amplitude parameters A^+ and A^-.

Triplet-based STDP Triplet-based STDP was developed by Pfister and Gerstner [105] to address the problem that pair-based STDP cannot account for an observed dependence on the frequency of spike pairs.

$$\Delta w = \begin{cases} \Delta w^{+} = \exp\left(\frac{-\Delta t_1}{\tau_{+}}\right)\left(A_2^{+} + A_3^{+}\exp\left(\frac{-\Delta t_2}{\tau_y}\right)\right) & \text{if } \Delta t = t_{post} \\ \Delta w^{-} = -\exp\left(\frac{-\Delta t_1}{\tau_{-}}\right)\left(A_2^{-} + A_3^{-}\exp\left(\frac{-\Delta t_3}{\tau_x}\right)\right) & \text{if } \Delta t = t_{pre}, \end{cases} \tag{8.28}$$

where A_2^{+}, A_2^{-}, A_3^{+}, and A_3^{-} are amplitude constants, $\Delta t_1 = t_{post} - t_{pre}$, $\Delta t_2 = t_{post(n)} - t_{post(n-1)} - \epsilon$ and $\Delta t_3 = t_{pre(n)} - t_{pre(n-1)} - \epsilon$ are time difference between the pre- and postsynaptic spikes. ϵ is typically a small constant which ensures that weight updates are applied at the correct time. Lastly, τ_{+}, τ_{-}, τ_x, τ_y are time constants.

Potentiation can occur when a post-synaptic spike is generated as a result of a two-step process. First a potentiation proportional to A_2^{+} for the current post-synaptic spike, and a potentiation proportional A_3^{+} for the interaction between the current and previous post-synaptic spike.

8.1.4 Supervised learning

There have been developed many supervised training algorithms for SNN, but none which works on all network architectures or without placing extensive restrictions on the neural code.

Algorithm	**Type**	**Architecture**
SpikeProp [106]	Backpropagation	Multilayer
Optimal Hebbain learning [107]	Hebbian	Conditional
ReSuMe [108]	ReSuMe	Single layer, multi-layer[109]
BPSL [110]	ReSuMe	Single layer
Reward modulated STDP [111]	Hebbian	Multi-layer

Table 8.1: Comparison of supervised learning algorithms for spiking neural networks.

SpikeProp

Developed by Bohte, Kok, and La Poutre [106], SpikeProp is gradient descent algorithm which to learn a set of desired firing times t_j^d at the postsynaptic neurons $j \in J$ for a set of input patterns $S^{in}(t)$. The algorithm is derived from the Spike Response Model (SRM) and required that all neurons are only allowed to fire one; subsequent spikes are to be ignored.

For SpikeProp, it is necessary for neurons to be initialized such that all neurons fire at least once for all patterns, to be able to calculate the gradient for the weights to that neuron. This makes it difficult to find good problem-independent weight initializations[112].

There has been developed an extension to SpikeProp by Booij and Nguyen [113] that allow input and hidden neurons to fire more than once.

Optimal Hebbian learning

Optimal Hebbian learning is a probabilistic approach that finds a likelihood which is a smooth function of the parameters by examining the pre- and postsynaptic firing times

[107]. This allows for the use of gradient descent based approaches to optimize the synaptic strengths.

ReSuMe

The ReSuMe learning rule uses a combination of Hebbian learning and gradient descent developed by Ponulak [108]. The rule is given by the equation:

$$\dot{w_{ik}}(t) = \left[S^d(t) - S^o(t)\right]\left[a \int_0^\infty W(s) s^{in}(t-s)ds\right], \tag{8.29}$$

where $S^d(t)$, $S^{in}(t)$ and $S^o(t)$ are the desired, pre- and postsynaptic spike trains respectively. The constant a represents the non-Hebbian contribution to the weight changes. $W(s)$ is the learning window of a time delay s:

$$W(s) = \begin{cases} +A_+ \cdot \exp(-s/\tau_+) & \text{if } s \geq 0, \\ -A_- \cdot \exp(s/\tau_)) & \text{if} s < 0, \end{cases} \tag{8.30}$$

with amplitudes $A_+, A_- \geq o$ and time constants $\tau_+, \tau_- > 0$.

The algorithm as given by Ponulak [108] can only be applied to networks without any hidden layers or as done in the original article, to learn readout neurons of an LSM. It has however, been shown experimentally that ReSuMe is unable to learn non-linear computations without hidden layers[109].

Sporea and Grüning [109] recently proposed a modification to ReSuMe which allows training of multilayer feedforward networks by making downstream neurons subject to multiplicative scaling. This modification could be used in conjunction with unrolling the network to train Recurrent Neural Network (RNN)s.

Biologically Plausible Supervised Learning (BPSL)

BPSL is a recent supervised learning algorithm by Taherkhani, Belatreche, Li, *et al.* [110], developed using recent insights into biological synaptic plasticity. While they do not demonstrate any higher accuracy of the learning result, the learning speed is much higher than with similar algorithms like ReSuMe. They report being able to train with ReSuMe in 3.5 seconds and 1 second for BPSL.

Reward-modulated synaptic plasticity

Reward-modulated synaptic plasticity is a proposal by Izhikevich [111] to solve the problem of how the firing pattern of neurons can be controlled far from the neuron exhibiting the wanted behavior. This can be accomplished by introducing some enzyme essential for plasticity of which there are several candidates, expressed here as the eligibility trace c:

$$\dot{c} = \frac{c}{\tau_c} + STDP(\tau) \cdot \delta(t - t_{pre}). \tag{8.31}$$

The change of synaptic strength can then be controlled by c and an extracellular dopamine level d:

$$\dot{w} = cd. \tag{8.32}$$

τ_c controls how fast c converges to 0.

8.1.5 Liquid state machines

LSM [114] is reservoir computing approach developed to elucidate the computational principals from biological neural microcircuits. LSMs use biologically plausible spiking neurons and builds the reservoir using biologically plausible topological constraints such as distant neurons being less likely to be connected. However, the biological plausibility of LSMs is challenged by the observation that it is not robust against damage in the network, such as missing or malfunctioning neurons[115].

Readout

There are two methods for solving practical problems with LSMs. The first is to use the state of the reservoir together with a general machine learning technique which then solves the problem. However, a more biologically plausible approach has been to use readout neurons that receive the spike trains of neurons in the reservoir. This was proposed already from the beginning [114]. Newer approaches improve on this by including Hebbian learning[123].

Learning

The reservoir as described by Maass, Natschläger, and Markram [114] is typically untrained. However, Hebbian learning implemented as Spike-timing-dependent plasticity has been shown to increase separation for speech data[19]. The authors were, however, uncertain about whether it would be more efficient to create a better liquid initially by adjusting the parameters. More recent attempts have found that this results in a significant increase in performance for complicated time series [124]. Similar approaches have been shown to work well with time-encoding based readout from the reservoir[125].

8.2 Proposed algorithm

There have been some limited attempts at using evolutionary methods to train SNNs. Some notable previous attempts include Pavlidis, Tasoulis, Plagianakos, *et al.* [126] which used a parallel version of the standard DE algorithm, and the work by Vazquez [127] which attempted to use the cuckoo search algorithm to train sets of individual spiking neurons to solve classification problems.

However, all of these approaches attempt to train simple, feedforward network structures that do not utilize the computation potential of SNN. To fully utilize these networks, the networks would need to be recurrent, that is, the synapses would need to form cycles. The only practically useful recurrent SNN architectures are LSM. While these have been shown to produce significantly better results when paired with unsupervised learning methods, the limitations of unsupervised learning will likely prevent them from becoming competitive against the state-of-the-art traditional ANNs, such as Long Short-Term Memory (LSTM)[128], [129].

The difficulty with training practically interesting SNNs architectures using black-box optimizers, is that the number of parameters quickly becomes greater than what the algorithms can efficiently handle. This thesis therefore proposes a specialized version of DE that utilizes synaptic plasticity and selection based on neuronal activity to make the search more efficient. The proposed algorithm is based on a simplified version of the improved LSHADE-EpSin algorithm resulting from the experiment in Chapter 7. The proposed algorithm is outlined in Algorithm 7.

8.2.1 Synaptic plasticity

Synaptic plasticity can function as a local search. However, synaptic plasticity would have to be used in a controlled manner, since training every individual using synaptic plasticity could cause the population to lose diversity and get stuck in local optima. Additionally, using synaptic plasticity would effectively require the computation time equaling one extra evaluation, two if the performance of the resulting network was needed; one to simulate the network with plasticity, and one to test afterwards.

The different types of synaptic plasticity discussed in Section 8.1.3 exhibit different behaviors, but the Triplet-Based STDP [105] has been shown to be more general than BCM[130].

The solution proposed here is to associate every individual $\mathbf{x_i^{(t)}}$ with a *synaptic individual* $\mathbf{s_i^{(t)}}$ that has been trained using synaptic plasticity. The synaptic individual is initialized such that $\mathbf{x_i^{(1)}} = \mathbf{s_i^{(1)}}$. The algorithm can then choose probabilistically, or based on the search progression, or a combination of the two, to update the synaptic individual by running a simulation. This approach does not require the fitness of the synaptic individual, and would, therefore, not need more time than approximately one additional fitness evaluation whenever the synaptic individual is updated.

Synaptic plasticity might only be useful toward the end of the search. To simplify the logic of this, the algorithm keeps $\mathbf{x_i^{(t)}} = \mathbf{s_i^{(t)}}$ until the first time the synaptic individual i is updated.

8.2.2 Current-to-pbest-synaptic mutation operator

This section propose including the synaptic individual into a new mutation operator called Current-to-pbest-synaptic, based on the current-to-pbest mutation operator described in Section 6.1.1, as follows:

$$\mathbf{v_i^{(t)}} = \mathbf{x_i^{(t)}} + F\left(\mathbf{x_{pbest}^{(t)}} - \mathbf{x_i^{(t)}}\right) + F\left(\mathbf{s_i^{(1)}} - \mathbf{x_i^{(t)}}\right) + F\left(\mathbf{x_{R_1}^{(t)}} - \mathbf{x_{R_2}^{(t)}}\right) \tag{8.33}$$

The mutation guides the search toward values found by training the network using synaptic plasticity. The operator is illustrated in Figure 8.2.

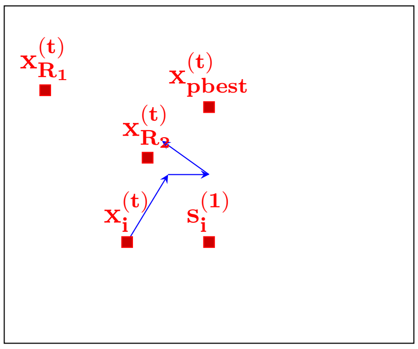


Figure 8.2: Illustration of the Current-to-pbest-synaptic mutation operator

8.2.3 Neuronal activity selection

The selection of individuals for the population might need to be modified for an efficient search, especially early in the search. Many weight configurations would result in either too few, or no neurons firing, and these individuals might provide too poor performance to properly guide the search.

A potential solution to this is to have a secondary condition to determine when an individual is better based on the neuronal activity as follows:

$$\mathbf{x_i}^{(t+1)} = \begin{cases} \mathbf{u_i} & \text{if } f\left(\mathbf{u_i^{(t)}}\right) < f\left(\mathbf{x_i^{(t)}}\right), \\ \mathbf{u_i} & \text{else if } f\left(\mathbf{u_i^{(t)}}\right) = f\left(\mathbf{x_i^{(t)}}\right) \wedge a\left(\mathbf{u_i^{(t)}}\right) > a\left(\mathbf{x_i^{(t)}}\right), \\ \mathbf{x_i}^{(t+1)} & \text{otherwise.} \end{cases} \tag{8.34}$$

Here, $a(\cdot)$ determines how many neurons fired.

8.2.4 Using the algorithm

The algorithm here can be used to train most types of SNNs, the only requirement being that enough networks can be evaluated within a reasonable time-frame, and that the performance of the network can be measured as a single real number. Tuning LSMs is therefore tempting, as these have been shown to achieve better performance when some training of the reservoir has taken place[124].

8.3 Conclusion

This chapter discussed SNN and proposed a new, specialized version of DE for training these networks. The new algorithm introduces the concept of a synaptic individual, which stores the parameters of a network that has been trained using synaptic plasticity. These parameters are used to guide the search through the new Current-to-pbest-synaptic mutation operator.

While it is impossible to know how well this new algorithm performs without being used in practice to train SNNs, it provides novel ideas that could work by itself, or be built on in future work.

Algorithm 7 The Synaptic LSHADE algorithm

```
$P^{(1)} \leftarrow \{\mathbf{x_i^{(1)}}, \mathbf{x_i^{(1)}}, \ldots, \mathbf{x_{N_p}^{(1)}}\}$                ▷ see Section 3.1 for details.
$Y^{(1)} \leftarrow \{\mathbf{x_i^{(1)}}, \mathbf{x_i^{(1)}}, \ldots, \mathbf{x_{N_p}^{(1)}}\}$       ▷ $Y^{(t)}$ is the synaptic individuals at generation $t$
$Y^{(1)}_{updated,i} \leftarrow 0$ for all $i$
$t \leftarrow 1$
$fes \leftarrow N_p$
Initialize memories $M_{CR}^{(t)}$, $M_F^{(t)}$ and $M_{freq}^{(t)}$ to 0.5.        ▷ see Section 7.1.2 for details.
while Termination criteria not met do
    $S_{CR} \leftarrow \emptyset, S_F \leftarrow \emptyset, S_{freq} \leftarrow \emptyset$
    $progression \leftarrow \frac{fes}{fex_{max}}$
    for $i \leftarrow 1, N_p$ do
        $r = rand(1, H)$ 1
        if $progression < \tanh(\tanh(0.5))$ then
            $F_i^{(t)} = 0.5$        ▷ Simplified version of the improvement found in Chapter 7
        else
            $F_i^{(t)} = randc(M_{F,r}^{(t)}, 0.1)$
        end if
        if $M_{CR,r}^{(t)} = \perp$ then
            $CR_i^{(t)} = 0$
        else
            $CR_i^{(t)} = randn(M_{CR,r}^{(t)}, 0.1)$
        end if
        $\mathbf{v_i^{(t)}} \leftarrow mutate(\mathbf{x_i^{(t)}})$                ▷ see Equation (8.33) for details
        $\mathbf{u_i^{(t)}} \leftarrow binaryCrossover(\mathbf{x_i^{(t)}}, \mathbf{v_i^{(t)}})$          ▷ see Section 3.3 for details.
        $fes \leftarrow fes + 1$
        if $progression > 0.75 \wedge rand(0, 1) < 0.5 \wedge (fes_{max} - fes) \geq 1$ then
            $s_i^{(t)} \leftarrow$ network trained using synaptic plasticity
            $fes \leftarrow fes + 1$
        end if
        if $f(\mathbf{u_i^{(t)}}) < f(\mathbf{x_i^{(t)}})$ then
            Insert $\mathbf{u_i^{(t)}}$ into $P^{(t+1)}$
            Insert $\mathbf{x_i^{(t)}}$ into archive                ▷ See Section 6.1.2 for details
        else
            if $f(\mathbf{u_i^{(t)}}) = f(\mathbf{x_i^{(t)}}) \wedge a(\mathbf{u_i^{(t)}}) > a(\mathbf{x_i^{(t)}})$ then
                Insert $\mathbf{u_i^{(t)}}$ into $P^{(t+1)}$
                Insert $\mathbf{x_i^{(t)}}$ into archive            ▷ See Section 6.1.2 for details
            else
                Insert $\mathbf{x_i^{(t)}}$ into $P^{(t+1)}$
            end if
        end if
    end for
    Reduce population size                                  ▷ See Section 7.1.3 for details
    Update $M_{CR,r}$, $M_{F,r}$, and $M_{freq,r}$.                ▷ see Section 7.1.2 for details
    if $N_p < 20$ for the first time then
        Execute local search as described in Section 7.1.3
    end if
    $t \leftarrow t + 1$
end while
```

Chapter 9

Further work

There is much further work that could be done to increase the chance of finding larger improvements. This section discusses some of it.

9.1 Standard differential evolution

It is likely improving the standard Differential Evolution algorithm using ADATE would succeed if using the improved grading of candidate programs used when improving CDE or LSHADE-EpSin. The main reason this was not done as part of this project is the difficulty of justifying using a large amount of computation resources on an algorithm that is unlikely to produce state-of-the-art results. However, due to being available in commercial and academic software packages, several of which have a configurable mutation operator. A significantly better mutation could potentially be adopted, and used by a larger community of users earlier than state-of-the-art variants.

9.2 Competitive differential evolution

The problem discovered with CDE was the faulty mechanics governing the competition among strategies, and the priority should be fixing it.

The likely best option is to change the way n_h is updated to take into account how large improvement each success has resulted in by updating n_h using the normalized improvement:

$$n_h^{(t+1)} = n_h^{(t)} + \frac{\Delta_h}{\sum_{h=1}^{H} \Delta_h} \tag{9.1}$$

with Δ_h being the sum of all improvements in the current generation for strategy h:

$$\Delta_h = \sum_{i \in D_h} \left| f\left(\mathbf{x}_\mathbf{i}^{(\mathbf{t})}\right) - f\left(\mathbf{u}_\mathbf{i}^{(\mathbf{t})}\right) \right| . \tag{9.2}$$

Here D_h is the set of all the indexes for individuals improved by strategy h.

This solution could however be affected by another problem with the competition mechanics, specifically that the adaptation slows down after several updates accumulate in $n_h^{(t+1)}$. Potentially to the point where it becomes unable to react to changes in the landscape. This can be solved using the a method similar to the adaptation mechanics for

F and CR in JADE[48], specifically, introducing a control parameter c into Equation (9.1):

$$n_h^{(t+1)} = (1-c)n_h^{(t)} + c\left(\frac{\Delta_h}{\sum_{h=1}^{H}\Delta_h}\right) \tag{9.3}$$

Using this update rule, the c could be changed to control how quickly n_h would be allowed to change.

The way method used to include the crossover rate in the strategies is rather coarse compared to other adaptation methods such as in SHADE. Using a version of CDE which includes these better methods would be preferable. One such version was entered into the CEC 2016 competition, and with the above modifications together with further improvement by ADATE, could be highly competitive[131].

9.3 LSHADE-EpSin

Due to the entire sinusoidal ensemble being shown to be unnecessary, including other parts of the algorithm into f would be the next step. Some candidates are:

- The weighted Lehmer mean represented in a way which would allow ADATE to produce independent heuristics for the crossover rate and scaling factor F.
- The Cauchy distribution and normal distribution used to sample the scaling factors and crossover rate respectively.

In practical contexts, it is often beneficial if the optimum is found quickly. In the grading used in this thesis, this is only weekly accounted for by using three steps of increasing fitness evaluation budgets. However, this could be made used as a larger requirement for candidate programs by incorporating how many function evaluations were used to the grade or adding by another grade which ADATE could use simultaneously.

9.4 Optimizing for specialized use-cases

Specialized versions like the one discussed in Chapter 8 could be created for other niche problems. However, it would be interesting to investigate the use of ADATE to automatically design such versions. The specialization to the low number of generations in Chapter 5 shows ADATE is capable.

The Ocean glider path planning problem mentioned in Section 3.7.2 required a power efficient algorithm. While the DE algorithm is inherently power efficient, the efficiency is dependent on the number of fitness function evaluations necessary to reach a good enough solution. As shown in Chapter 5, the mutation operator has a large effect on the performance of the algorithm and how quickly it converges. While the intention in that project was to use DE with a larger number of evaluations later, the observations can be used to motivate experiments targeting niche problems.

Chapter 10

Conclusions

This thesis had three goals. First (RQ1), to understand how synthesized modifications to Differential Evolution might be evaluated quickly enough for ADATE. Second (RQ2), to determine which parts of the Differential Evolution algorithm might be improved, and third (RQ3), to understand how specialized versions of Differential Evolution for training Spiking Neural Networks might be created.

Three experiments, discussed in Chapters 5 to 7, were used to answer RQ1. In all three experiments, the candidate programs were evaluated using synthetic benchmark functions, as these can be evaluated much more quickly than landscape generators[82]. Additionally, synthetic benchmark functions are often designed to study optimization on problems with special characteristics. As such, the landscapes are often better defined than machine learning models, making it easier to create representative problem sets.

To ensure all problems contributed equally to the grade, Chapter 5 proposed an approach based on the ratio of the achieved fitness on a problem to the median of the fitness achieved by the original algorithm. This was changed in Chapter 6 to a method based on the approximate cumulative density function to accommodate problems that include both negative and positive fitness values.

The maximum number of allowed function evaluation was increased for each of the three experiments. The first experiment prioritized evaluating a high number of candidate programs by using a low number of training and validation problems, all with relatively few generations. Because of overfitting to the low number of generations, the number of generations was increased significantly for the second experiment. Additionally, the number of problems was increased to further enhance the statistical certainty of the evaluation. This resulted in the successful improvement of the pool of strategies in the Competitive Differential Evolution algorithm.

In the third, and last experiment, the statistical certainty of evaluations was further improved by using an even higher number of allowed function evaluations. This experiment succeeded in improving the mutation heuristics in the state-of-the-art LSHADE-EpSin algorithm resulting in an algorithm that is both smaller and achieves better results on problems with 30 dimensions. It is possible that larger improvements could be found if ADATE had been able to test more programs. This could be achieved by either using fewer repetitions of each problem or using more computing time for the experiment. Nevertheless, this shows the evaluation both provides enough guidance, and enables the evaluation of candidate programs quickly enough, for the ADATE search.

To answer RQ2, Chapter 5 included a discussion of which parts to improve. That discussion resulted in the parts selected for improvement were related to the mutation heuristics for all three experiments. Improving the mutation had several benefits; namely, since the implementation of all mutation operators has a similar overall algorithmic structure, the number of transformations that would be necessary to transform one into another, provided all the required background knowledge was made available, is minimal. Additionally, the mutation is at the core of the algorithm, with different mutation operators changing the performance of the search dramatically, as evidenced by the results in Tables 5.5 to 5.8. This means that even simple modifications could yield significant improvement of the algorithms.

The successful improvement of both the Competitive Differential Evolution and LSHADE-EpSin supports the conclusion that this was a good choice. However, there are other parts of the algorithms that might also yield improvements, especially concerning the adaptation mechanics in LSHADE-EpSin.

Creating specialized variations of Differential Evolution for training spiking neural networks to answer RQ3, was limited to theoretical discussions. The major problem preventing the use of Differential Evolution to train spiking neural networks is the difficulty of optimizing such a high-dimensionality problem using black-box methods. However, specialized algorithms can use knowledge about the model being optimized to guide the search more effectively. This was used to motivate the proposal of a specialized version of Differential Evolution in Chapter 8 that incorporates synaptic plasticity into the search, and a new mechanism for selecting the new generation that uses the neuronal activity as a secondary measure.

While it is impossible to know how well this algorithm will perform in real-life without it being tested on practical problems, it does show how training methods and information specific to spiking neural networks can be included in the mutation and selection mechanics of Differential Evolution in a manner that might make the search more efficient. Although this is not the first attempt at creating a specialized version of Differential Evolution, it is the first version intended to train spiking neural networks of which we are aware.

Bibliography

[1] F. Glover, “Tabu searchpart i,” *ORSA Journal on computing*, vol. 1, no. 3, pp. 190–206, 1989.

[2] S. Kirkpatrick, C. D. Gelatt, M. P. Vecchi, *et al.*, “Optimization by simulated annealing,” *Science*, vol. 220, no. 4598, pp. 671–680, 1983.

[3] D. Karaboga, “An idea based on honey bee swarm for numerical optimization,” Technical report-tr06, Erciyes university, engineering faculty, computer engineering department, Tech. Rep., 2005.

[4] R. Eberhart and J. Kennedy, “A new optimizer using particle swarm theory,” in *Micro Machine and Human Science, 1995. MHS’95., Proceedings of the Sixth International Symposium on*, IEEE, 1995, pp. 39–43.

[5] J. H. Holland, *Adaptation in natural and artificial systems: An introductory analysis with applications to biology, control, and artificial intelligence.* MIT press, 1992.

[6] H.-G. Beyer and H.-P. Schwefel, “Evolution strategies–a comprehensive introduction,” *Natural computing*, vol. 1, no. 1, pp. 3–52, 2002.

[7] R. Storn and K. Price, *Differential evolution-a simple and efficient adaptive scheme for global optimization over continuous spaces.* ICSI Berkeley, 1995, vol. 3.

[8] M. Vasile, E. Minisci, and M. Locatelli, “An inflationary differential evolution algorithm for space trajectory optimization,” *IEEE Transactions on Evolutionary Computation*, vol. 15, no. 2, pp. 267–281, 2011.

[9] A. Zamuda and J. D. H. Sosa, “Differential evolution and underwater glider path planning applied to the short-term opportunistic sampling of dynamic mesoscale ocean structures,” *Applied Soft Computing*, vol. 24, pp. 95–108, 2014.

[10] L. Jebaraj, C. Venkatesan, I. Soubache, and C. C. A. Rajan, “Application of differential evolution algorithm in static and dynamic economic or emission dispatch problem: A review,” *Renewable and Sustainable Energy Reviews*, 2017.

[11] R. Olsson, “Inductive functional programming using incremental program transformation; and, execution of logic programs by iterative-deepening a* sld-tree search,” PhD thesis, University of Oslo, Department of Informatics, 1994.

[12] ——, “Inductive functional programming using incremental program transformation,” *Artificial intelligence*, vol. 74, no. 1, pp. 55–81, 1995.

[13] J. R. Olsson, “How to invent functions,” in *Genetic Programming*, Springer, 1999, pp. 232–243.

[14] A. Løkketangen and R. Olsson, “Generating meta-heuristic optimization code using adate,” *Journal of Heuristics*, vol. 16, no. 6, pp. 911–930, 2010.

[15] K. Larsen, L. V. Magnusson, and R. Olsson, "Edge pixel classification using automatic programming.," in *NIK*, 2014.

[16] H. Vu and R. Olsson, "Automatic improvement of graph based image segmentation," *Advances in Visual Computing*, pp. 578–587, 2012.

[17] P. A. Merolla, J. V. Arthur, R. Alvarez-Icaza, A. S. Cassidy, J. Sawada, F. Akopyan, B. L. Jackson, N. Imam, C. Guo, Y. Nakamura, *et al.*, "A million spiking-neuron integrated circuit with a scalable communication network and interface," *Science*, vol. 345, no. 6197, pp. 668–673, 2014.

[18] W. Maass, "Networks of spiking neurons: The third generation of neural network models," *Neural networks*, vol. 10, no. 9, pp. 1659–1671, 1997.

[19] D. Norton and D. A. Ventura, "Preparing more effective liquid state machines using hebbian learning," 2006.

[20] P. Bujok, J. Tvrdk, and R. Polakova, "Differential evolution with rotation-invariant mutation and competing-strategies adaptation," in *Evolutionary Computation (CEC), 2014 IEEE Congress on*, IEEE, 2014, pp. 2253–2258.

[21] N. H. Awad, M. Z. Ali, P. N. Suganthan, and R. G. Reynolds, "An ensemble sinusoidal parameter adaptation incorporated with l-shade for solving cec2014 benchmark problems," in *Evolutionary Computation (CEC), 2016 IEEE Congress on*, IEEE, 2016, pp. 2958–2965.

[22] F. Marini and B. Walczak, "Particle swarm optimization (pso). a tutorial," *Chemometrics and Intelligent Laboratory Systems*, vol. 149, pp. 153–165, 2015.

[23] J. Liang, B. Qu, and P. Suganthan, "Problem definitions and evaluation criteria for the cec 2014 special session and competition on single objective real-parameter numerical optimization," *Computational Intelligence Laboratory, Zhengzhou University, Zhengzhou China and Technical Report, Nanyang Technological University, Singapore*, 2013.

[24] T. P. Hettmansperger and J. W. McKean, *Robust nonparametric statistical methods.* CRC Press, 2010.

[25] R. Storn, "On the usage of differential evolution for function optimization," in *Fuzzy Information Processing Society, 1996. NAFIPS., 1996 Biennial Conference of the North American*, IEEE, 1996, pp. 519–523.

[26] K. Price, R. M. Storn, and J. A. Lampinen, *Differential evolution: A practical approach to global optimization.* Springer Science & Business Media, 2006.

[27] E. Mezura-Montes, J. Velázquez-Reyes, and C. A. Coello Coello, "A comparative study of differential evolution variants for global optimization," in *Proceedings of the 8th annual conference on Genetic and evolutionary computation*, ACM, 2006, pp. 485–492.

[28] P. Kaelo and M. Ali, "A numerical study of some modified differential evolution algorithms," *European journal of operational research*, vol. 169, no. 3, pp. 1176–1184, 2006.

[29] H.-Y. Fan and J. Lampinen, "A trigonometric mutation operation to differential evolution," *Journal of global optimization*, vol. 27, no. 1, pp. 105–129, 2003.

[30] R. Salomon, “Re-evaluating genetic algorithm performance under coordinate rotation of benchmark functions. a survey of some theoretical and practical aspects of genetic algorithms,” *BioSystems*, vol. 39, no. 3, pp. 263–278, 1996.

[31] D. Zaharie, “Critical values for the control parameters of differential evolution algorithms,” in *Proceedings of MENDEL*, vol. 2, 2002, p. 6267.

[32] ——, “A comparative analysis of crossover variants in differential evolution,” *Proceedings of IMCSIT*, vol. 2007, pp. 171–181, 2007.

[33] S. Kukkonen and J. Lampinen, “An empirical study of control parameters for the third version of generalized differential evolution (gde3),” in *Evolutionary Computation, 2006. CEC 2006. IEEE Congress on*, IEEE, 2006, pp. 2002–2009.

[34] S. Kukkonen *et al.*, “Generalized differential evolution for global multi-objective optimization with constraints,” *Acta Universitatis Lappeenrantaensis*, 2012.

[35] S. Das and P. N. Suganthan, “Differential evolution: A survey of the state-of-the-art,” *IEEE transactions on evolutionary computation*, vol. 15, no. 1, pp. 4–31, 2011.

[36] R. Tanabe and A. S. Fukunaga, “Improving the search performance of shade using linear population size reduction,” in *Evolutionary Computation (CEC), 2014 IEEE Congress on*, IEEE, 2014, pp. 1658–1665.

[37] G. Harik, E. Cantú-Paz, D. E. Goldberg, and B. L. Miller, “The gambler’s ruin problem, genetic algorithms, and the sizing of populations,” *Evolutionary Computation*, vol. 7, no. 3, pp. 231–253, 1999.

[38] C. R. Reeves, “Using genetic algorithms with small populations.,” in *ICGA*, vol. 590, 1993, p. 92.

[39] D. E. Goldberg, K. Sastry, and T. Latoza, “On the supply of building blocks,” in *Proceedings of the 3rd Annual Conference on Genetic and Evolutionary Computation*, Morgan Kaufmann Publishers Inc., 2001, pp. 336–342.

[40] D. E. Goldberg, K. Deb, and J. H. Clark, “Genetic algorithms, noise, and the sizing of populations,” *Urbana*, vol. 51, p. 61 801, 1991.

[41] F. G. Lobo, “Idealized dynamic population sizing for uniformly scaled problems,” in *Proceedings of the 13th annual conference on Genetic and evolutionary computation*, ACM, 2011, pp. 917–924.

[42] V. K. Koumousis and C. P. Katsaras, “A saw-tooth genetic algorithm combining the effects of variable population size and reinitialization to enhance performance,” *IEEE Transactions on Evolutionary Computation*, vol. 10, no. 1, pp. 19–28, 2006.

[43] T. Hu, S. Harding, and W. Banzhaf, “Variable population size and evolution acceleration: A case study with a parallel evolutionary algorithm,” *Genetic Programming and Evolvable Machines*, vol. 11, no. 2, pp. 205–225, 2010.

[44] G. R. Harik and F. G. Lobo, “A parameter-less genetic algorithm,” in *Proceedings of the 1st Annual Conference on Genetic and Evolutionary Computation-Volume 1*, Morgan Kaufmann Publishers Inc., 1999, pp. 258–265.

[45] J. C. Costa, R. Tavares, and A. Rosa, “An experimental study on dynamic random variation of population size,” in *Systems, Man, and Cybernetics, 1999. IEEE SMC’99 Conference Proceedings. 1999 IEEE International Conference on*, IEEE, vol. 1, 1999, pp. 607–612.

[46] F. G. Lobo and C. F. Lima, “Adaptive population sizing schemes in genetic algorithms,” in *Parameter Setting in Evolutionary Algorithms*, Springer, 2007, pp. 185–204.

[47] A. K. Qin and P. N. Suganthan, “Self-adaptive differential evolution algorithm for numerical optimization,” in *Evolutionary Computation, 2005. The 2005 IEEE Congress on*, IEEE, vol. 2, 2005, pp. 1785–1791.

[48] J. Zhang and A. C. Sanderson, “Jade: Adaptive differential evolution with optional external archive,” *IEEE Transactions on evolutionary computation*, vol. 13, no. 5, pp. 945–958, 2009.

[49] J. Tvrdk, “Competitive differential evolution,” in *MENDEL*, 2006, pp. 7–12.

[50] R. Mallipeddi, P. N. Suganthan, Q.-K. Pan, and M. F. Tasgetiren, “Differential evolution algorithm with ensemble of parameters and mutation strategies,” *Applied Soft Computing*, vol. 11, no. 2, pp. 1679–1696, 2011.

[51] P.-c. Wang, X. Qian, and X.-h. Hu, “A novel differential evolution algorithm based on chaos local search,” in *Information Engineering and Computer Science, 2009. ICIECS 2009. International Conference on*, IEEE, 2009, pp. 1–4.

[52] D. Jia, G. Zheng, and M. K. Khan, “An effective memetic differential evolution algorithm based on chaotic local search,” *Information Sciences*, vol. 181, no. 15, pp. 3175–3187, 2011.

[53] I. Poikolainen and F. Neri, “Differential evolution with concurrent fitness based local search,” in *Evolutionary Computation (CEC), 2013 IEEE Congress on*, IEEE, 2013, pp. 384–391.

[54] M. Ali, M. Pant, and A. Nagar, “Two local search strategies for differential evolution,” in *Bio-Inspired Computing: Theories and Applications (BIC-TA), 2010 IEEE Fifth International Conference on*, IEEE, 2010, pp. 1429–1435.

[55] H. Peng and Z. Wu, “Heterozygous differential evolution with taguchi local search,” *Soft Computing*, vol. 19, no. 11, pp. 3273–3291, 2015.

[56] S. Das, S. S. Mullick, and P. N. Suganthan, “Recent advances in differential evolution–an updated survey,” *Swarm and Evolutionary Computation*, vol. 27, pp. 1–30, 2016.

[57] P. D. Summers, “A methodology for lisp program construction from examples,” *Journal of the ACM (JACM)*, vol. 24, no. 1, pp. 161–175, 1977.

[58] S. Muggleton and C. Feng, “Efficient Induction of Logic Programs,” in *Algorithmic Learning Theory*, 1990, pp. 368–381.

[59] S. Muggleton and L. De Raedt, “Inductive logic programming: Theory and methods,” *The Journal of Logic Programming*, vol. 19, pp. 629–679, 1994.

[60] S. Muggleton, “Inverse entailment and progol,” *New Generation Computing*, vol. 13, pp. 245–286, 3&4 1995. DOI: `10.1007/BF03037227`.

[61] P. Flener, “Inductive logic program synthesis with dialogs,” in *International Conference on Inductive Logic Programming*, Springer, 1996, pp. 175–198.

[62] M. Hofmann, E. Kitzelmann, and U. Schmid, “A unifying framework for analysis and evaluation of inductive programming systems,” in *Proceedings of the Second Conference on Artificial General Intelligence*, Citeseer, 2009, pp. 55–60.

[63] U. Schmid and E. Kitzelmann, "Inductive rule learning on the knowledge level," *Cognitive Systems Research*, vol. 12, no. 3, pp. 237–248, 2011.

[64] E. Kitzelmann, U. Schmid, M. Mühlpfordt, and F. Wysotzki, "Inductive synthesis of functional programs," in *Artificial intelligence, automated reasoning, and symbolic computation*, Springer, 2002, pp. 26–37.

[65] S. Katayama, "Magichaskeller: System demonstration," in *Proceedings of AAIP 2011 4th International Workshop on Approaches and Applications of Inductive Programming*, 2011, p. 63.

[66] ——, "Recent improvements of magichaskeller," in *International Workshop on Approaches and Applications of Inductive Programming*, Springer, 2009, pp. 174–193.

[67] ——, "Towards human-level inductive functional programming," in *International Conference on Artificial General Intelligence*, Springer, 2015, pp. 111–120.

[68] J. R. Koza, *Genetic programming: On the programming of computers by means of natural selection.* MIT press, 1992, vol. 1.

[69] R. Poli, W. B. Langdon, N. F. McPhee, and J. R. Koza, *A field guide to genetic programming.* Lulu. com, 2008.

[70] R. Hrbacek and V. Dvorak, "Bent function synthesis by means of cartesian genetic programming," in *International Conference on Parallel Problem Solving from Nature*, Springer, 2014, pp. 414–423.

[71] L. V. Magnusson and R. Olsson, "Improving the canny edge detector using automatic programming: Improving the filter," in *Image, Vision and Computing (ICIVC), International Conference on*, IEEE, 2016, pp. 36–40.

[72] ——, "Improving the canny edge detector using automatic programming: Improving hysteresis thresholding," *Norsk Informatikkonferanse (NIK)*, 2016.

[73] ——, "Improving the canny edge detector using automatic programming: Improving non-max suppression," in *Proceedings of the 2016 on Genetic and Evolutionary Computation Conference*, ACM, 2016, pp. 461–468.

[74] E. K. Burke, M. Hyde, G. Kendall, and J. Woodward, "A genetic programming hyper-heuristic approach for evolving 2-d strip packing heuristics," *IEEE Transactions on Evolutionary Computation*, vol. 14, no. 6, pp. 942–958, 2010.

[75] E. Burke, M. Hyde, and G. Kendall, "Evolving bin packing heuristics with genetic programming," *Parallel Problem Solving from Nature-PPSN IX*, pp. 860–869, 2006.

[76] S. Allen, E. K. Burke, M. Hyde, and G. Kendall, "Evolving reusable 3d packing heuristics with genetic programming," in *Proceedings of the 11th Annual conference on Genetic and evolutionary computation*, ACM, 2009, pp. 931–938.

[77] D. Jakobovi and K. Marasovi, "Evolving priority scheduling heuristics with genetic programming," *Applied Soft Computing*, vol. 12, no. 9, pp. 2781–2789, 2012.

[78] A. S. Fukunaga, "Evolving local search heuristics for sat using genetic programming," in *Genetic and Evolutionary Computation Conference*, Springer, 2004, pp. 483–494.

[79] M. Bader-El-Den and R. Poli, "Generating sat local-search heuristics using a gp hyper-heuristic framework," in *International Conference on Artificial Evolution (Evolution Artificielle)*, Springer, 2007, pp. 37–49.

[80] A. K. Qin and X. Li, “Differential evolution on the cec-2013 single-objective continuous optimization testbed,” in *Evolutionary Computation (CEC), 2013 IEEE Congress on*, IEEE, 2013, pp. 1099–1106.

[81] J. Liang, B. Qu, P. Suganthan, and A. G. Hernández-Daz, “Problem definitions and evaluation criteria for the cec 2013 special session on real-parameter optimization,” *Computational Intelligence Laboratory, Zhengzhou University, Zhengzhou, China and Nanyang Technological University, Singapore, Technical Report*, vol. 201212, 2013.

[82] M. Gallagher and B. Yuan, “A general-purpose tunable landscape generator,” *IEEE transactions on evolutionary computation*, vol. 10, no. 5, pp. 590–603, 2006.

[83] J. Tvrdk, L. Misik, and I. Krivy, “Competing heuristics in evolutionary algorithms,” *Intell. Technol. Theory Applicat*, pp. 159–165, 2002.

[84] J. Tvrdk, “Differential evolution with competitive setting of control parameters,” *Task Quarterly*, vol. 11, no. 1-2, pp. 169–179, 2007.

[85] ——, “Adaptation in differential evolution: A numerical comparison,” *Applied Soft Computing*, vol. 9, no. 3, pp. 1149–1155, 2009.

[86] J. Tvrdk and R. Poláková, “Competitive differential evolution applied to cec 2013 problems,” in *Evolutionary Computation (CEC), 2013 IEEE Congress on*, IEEE, 2013, pp. 1651–1657.

[87] R. Poláková, J. Tvrdk, and P. Bujok, “Controlled restart in differential evolution applied to cec2014 benchmark functions,” in *Evolutionary Computation (CEC), 2014 IEEE Congress on*, IEEE, 2014, pp. 2230–2236.

[88] J. Tvrdk and R. Poláková, “Competitive differential evolution for constrained problems,” in *Evolutionary Computation (CEC), 2010 IEEE Congress on*, IEEE, 2010, pp. 1–8.

[89] R. Tanabe and A. Fukunaga, “Success-history based parameter adaptation for differential evolution,” in *2013 IEEE Congress on Evolutionary Computation*, Jun. 2013, pp. 71–78. DOI: 10.1109/CEC.2013.6557555.

[90] E. M. Izhikevich, “Which model to use for cortical spiking neurons?” *IEEE transactions on neural networks*, vol. 15, no. 5, pp. 1063–1070, 2004.

[91] A. L. Hodgkin and A. F. Huxley, “A quantitative description of membrane current and its application to conduction and excitation in nerve,” *The Journal of physiology*, vol. 117, no. 4, p. 500, 1952.

[92] R. FitzHugh, “Impulses and physiological states in theoretical models of nerve membrane,” *Biophysical journal*, vol. 1, no. 6, p. 445, 1961.

[93] C. Morris and H. Lecar, “Voltage oscillations in the barnacle giant muscle fiber.,” *Biophysical journal*, vol. 35, no. 1, p. 193, 1981.

[94] B. J. Grzyb, E. Chinellato, G. M. Wojcik, and W. A. Kaminski, “Which model to use for the liquid state machine?” In *2009 International Joint Conference on Neural Networks*, IEEE, 2009, pp. 1018–1024.

[95] W. Gerstner and W. M. Kistler, *Spiking neuron models: Single neurons, populations, plasticity.* Cambridge university press, 2002.

[96] E. M. Izhikevich, "Resonate-and-fire neurons," *Neural networks*, vol. 14, no. 6, pp. 883–894, 2001.

[97] J. Hindmarsh and R. Rose, "A model of neuronal bursting using three coupled first order differential equations," *Proceedings of the Royal Society of London B: Biological Sciences*, vol. 221, no. 1222, pp. 87–102, 1984.

[98] A. Citri and R. C. Malenka, "Synaptic plasticity: Multiple forms, functions, and mechanisms," *Neuropsychopharmacology*, vol. 33, no. 1, pp. 18–41, 2008.

[99] D. O. Hebb, *The organization of behavior: A neuropsychological approach.* John Wiley & Sons, 1949.

[100] P. Dayan and L. F. Abbott, *Theoretical neuroscience: Computational and mathematical modeling of neural systems*, 1st. The MIT Press, Dec. 2001, ISBN: 9780262041997. [Online]. Available: `http://amazon.com/o/ASIN/0262041995/`.

[101] C. H. Barlow, H. Barlow, and P. Fddik, "Adaptation and decorrelation," 1989.

[102] E. Oja, "Simplified neuron model as a principal component analyzer," *Journal of mathematical biology*, vol. 15, no. 3, pp. 267–273, 1982.

[103] E. L. Bienenstock, L. N. Cooper, and P. W. Munro, "Theory for the development of neuron selectivity: Orientation specificity and binocular interaction in visual cortex," *The Journal of Neuroscience*, vol. 2, no. 1, pp. 32–48, 1982.

[104] E. M. Izhikevich and N. S. Desai, "Relating stdp to bcm," *Neural computation*, vol. 15, no. 7, pp. 1511–1523, 2003.

[105] J.-P. Pfister and W. Gerstner, "Triplets of spikes in a model of spike timing-dependent plasticity," *The Journal of neuroscience*, vol. 26, no. 38, pp. 9673–9682, 2006.

[106] S. M. Bohte, J. N. Kok, and H. La Poutre, "Error-backpropagation in temporally encoded networks of spiking neurons," *Neurocomputing*, vol. 48, no. 1, pp. 17–37, 2002.

[107] J.-P. Pfister, D. Barber, and W. Gerstner, "Optimal hebbian learning: A probabilistic point of view," in *Artificial Neural Networks and Neural Information ProcessingICANN/ICONIP 2003*, Springer, 2003, pp. 92–98.

[108] F. Ponulak, "Resume-new supervised learning method for spiking neural networks," *Institute of Control and Information Engineering, Poznan University of Technology.(Available online at: Http://d1. cie. put. poznan. pl/˜ fp/research. html)*, 2005.

[109] I. Sporea and A. Grüning, "Supervised learning in multilayer spiking neural networks," *Neural computation*, vol. 25, no. 2, pp. 473–509, 2013.

[110] A. Taherkhani, A. Belatreche, Y. Li, and L. P. Maguire, "A new biologically plausible supervised learning method for spiking neurons.," in *ESANN*, 2014.

[111] E. M. Izhikevich, "Solving the distal reward problem through linkage of stdp and dopamine signaling," *Cerebral cortex*, vol. 17, no. 10, pp. 2443–2452, 2007.

[112] B. Schrauwen and J. Van Campenhout, "Extending spikeprop," in *Neural Networks, 2004. Proceedings. 2004 IEEE International Joint Conference on*, IEEE, vol. 1, 2004.

[113] O. Booij and H. tat Nguyen, “A gradient descent rule for spiking neurons emitting multiple spikes,” *Information Processing Letters*, vol. 95, no. 6, pp. 552–558, 2005.

[114] W. Maass, T. Natschläger, and H. Markram, “Real-time computing without stable states: A new framework for neural computation based on perturbations,” *Neural computation*, vol. 14, no. 11, pp. 2531–2560, 2002.

[115] H. Hazan and L. M. Manevitz, “Topological constraints and robustness in liquid state machines,” *Expert Systems with Applications*, vol. 39, no. 2, pp. 1597–1606, 2012.

[116] M. LukoEviIus and H. Jaeger, “Reservoir computing approaches to recurrent neural network training,” *Computer Science Review*, vol. 3, no. 3, pp. 127–149, 2009.

[117] E. Goodman and D. A. Ventura, “Spatiotemporal pattern recognition via liquid state machines,” 2006.

[118] D. Norton and D. Ventura, “Improving liquid state machines through iterative refinement of the reservoir,” *Neurocomputing*, vol. 73, no. 16, pp. 2893–2904, 2010.

[119] E. Hourdakis and P. Trahanias, “Improving the classification performance of liquid state machines based on the separation property,” in *Engineering Applications of Neural Networks*, Springer, 2011, pp. 52–62.

[120] G. M. Wojcik and M. Wany, “Bray-curtis metrics as measure of liquid state machine separation ability in function of connections density,” *Procedia Computer Science*, vol. 51, pp. 2979–2983, 2015.

[121] J. Chrol-Cannon and Y. Jin, “On the correlation between reservoir metrics and performance for time series classification under the influence of synaptic plasticity,” *PloS one*, vol. 9, no. 7, e101792, 2014.

[122] K. P. Dockendorf, I. Park, P. He, J. C. Prìncipe, and T. B. DeMarse, “Liquid state machines and cultured cortical networks: The separation property,” *Biosystems*, vol. 95, no. 2, pp. 90–97, 2009.

[123] Y. Zhang, P. Li, Y. Jin, and Y. Choe, “A digital liquid state machine with biologically inspired learning and its application to speech recognition,” *IEEE transactions on neural networks and learning systems*, vol. 26, no. 11, pp. 2635–2649, 2015.

[124] J. Chrol-Cannon and Y. Jin, “Learning structure of sensory inputs with synaptic plasticity leads to interference,” *Frontiers in computational neuroscience*, vol. 9, 2015.

[125] H. Paugam-Moisy, R. Martinez, and S. Bengio, “Delay learning and polychronization for reservoir computing,” *Neurocomputing*, vol. 71, no. 7, pp. 1143–1158, 2008.

[126] N. Pavlidis, O. Tasoulis, V. P. Plagianakos, G. Nikiforidis, and M. Vrahatis, “Spiking neural network training using evolutionary algorithms,” in *Neural Networks, 2005. IJCNN’05. Proceedings. 2005 IEEE International Joint Conference on*, IEEE, vol. 4, 2005, pp. 2190–2194.

[127] R. A. Vazquez, “Training spiking neural models using cuckoo search algorithm,” in *Evolutionary Computation (CEC), 2011 IEEE Congress on*, IEEE, 2011, pp. 679–686.

[128] K. Greff, R. K. Srivastava, J. Koutnik, B. R. Steunebrink, and J. Schmidhuber, “Lstm: A search space odyssey,” *ArXiv preprint arXiv:1503.04069*, 2015.

[129] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[130] J. Gjorgjieva, C. Clopath, J. Audet, and J.-P. Pfister, "A triplet spike-timing–dependent plasticity model generalizes the bienenstock–cooper–munro rule to higher-order spatiotemporal correlations," *Proceedings of the National Academy of Sciences*, vol. 108, no. 48, pp. 19 383–19 388, 2011.

[131] R. Poláková, J. Tvrdk, and P. Bujok, "Evaluating the performance of l-shade with competing strategies on cec2014 single parameter-operator test suite," in *Evolutionary Computation (CEC), 2016 IEEE Congress on*, IEEE, 2016, pp. 1181–1187.

[132] . G. Rou, C. Rdoi, A. Florescu, P. Guglielmi, and M. Pastorelli, "The analysis of the solutions for harmonic elimination pwm bipolar waveform with a specialized differential evolution algorithm," in *Optimization of Electrical and Electronic Equipment (OPTIM), 2012 13th International Conference on*, IEEE, 2012, pp. 814–821.

[133] M. Jamil and X.-S. Yang, "A literature survey of benchmark functions for global optimisation problems," *International Journal of Mathematical Modelling and Numerical Optimisation*, vol. 4, no. 2, pp. 150–194, 2013.

Appendix A

Optimization benchmark problems

This appendix describes the optimization benchmark functions used as part of this project. Table A.1 and A.2 list all basis functions, while some have been combined to form compound functions which is listed in table A.3.

A.1 Test function collections

Several good test function collections have been used to compile this document. The resources are listed in order of importance.

- M. Jamil and X.-S. Yang, "A literature survey of benchmark functions for global optimisation problems," *International Journal of Mathematical Modelling and Numerical Optimisation*, vol. 4, no. 2, pp. 150–194, 2013
- `http://al-roomi.org/benchmarks/unconstrained`
- `http://infinity77.net/global_optimization/test_functions.html`
- J. Liang, B. Qu, and P. Suganthan, "Problem definitions and evaluation criteria for the cec 2014 special session and competition on single objective real-parameter numerical optimization," *Computational Intelligence Laboratory, Zhengzhou University, Zhengzhou China and Technical Report, Nanyang Technological University, Singapore*, 2013
- J. Liang, B. Qu, P. Suganthan, *et al.*, "Problem definitions and evaluation criteria for the cec 2013 special session on real-parameter optimization," *Computational Intelligence Laboratory, Zhengzhou University, Zhengzhou, China and Nanyang Technological University, Singapore, Technical Report*, vol. 201212, 2013

A.2 Properties

Separability Functions where each variable can be optimized independently of the other variables.

Modality Multimodal functions have several peaks and can have multiple local optima which can cause the optimizer to get stuck in a local optima.

	Function name	**Dimensions**
f_1	Ackley's Function	n
f_2	Adjiman's Function	2
f_3	Alpine Function No.01	n
f_4	Alpine Function No.02	n
f_5	ANNs XOR Function	9
f_6	Bent cigar	n
f_7	Bird Function	2
f_8	Brad Function	3
f_9	Brown Function	n
f_{10}	Bukin's Function No.06	2
f_{11}	Cosine Mixture Function	n
f_{12}	Cross-In Tray Function	2
f_{13}	Crowned Cross Function	2
f_{14}	Davis' Function	2
f_{15}	Deflected Corrugated Spring Function	n
f_{16}	Discus Function	n
f_{17}	Downhill Step Function	2
f_{18}	Drop-Wave Function	2
f_{19}	Egg Crate Function	2
f_{20}	Egg-Holder Function	n
f_{21}	Giunta's Function	n
f_{22}	Griewank	n
f_{23}	HappyCat Function	n
f_{24}	HGBat Function	n
f_{25}	High Conditioned Elliptic Function	n
f_{26}	Hosaki's Function	2
f_{27}	Katsuura Function	n
f_{28}	Leon's Function	2
f_{29}	L (or F2) Function	n
f_{30}	Lunacek's bi-Rastrigin Function	n
f_{31}	Mishra's Function No.03	2
f_{32}	Modified Schaffer's Function No.01	2
f_{33}	Modified Schwefel function	n
f_{34}	Pathological Function	n
f_{35}	Paviani	10
f_{36}	Peaks function	2
f_{37}	Powell sum	n
f_{38}	Price's Function No.02	2
f_{39}	Qing's Function	n
f_{40}	Quintic Function	n
f_{41}	Rastrigin function	n
f_{42}	Rosenbrocks Function	n
f_{43}	Salomon's Function	n
f_{44}	Sawtoothxy Function	2

Table A.1: Base optimization functions - part 1

	Function name	**Dimensions**
f_{45}	Schaffer's F6	2
f_{46}	Schwefel	n
f_{47}	Schwefel F2.21	n
f_{48}	Schwefel F2.26	n
f_{49}	Shubert Function	n
f_{50}	Shubert 3 Function	n
f_{51}	Shubert 4 Function	n
f_{52}	Six-Hump Camel-Back Function	2
f_{53}	Sphere function	n
f_{54}	Step Function No.02	n
f_{55}	Styblinski-Tang	n
f_{56}	Tsoulos' Function	2
f_{57}	Ursem Function No.03	2
f_{58}	Ursem-Waves Function	2
f_{59}	Venter and Sobiezcczanski-Sobieski's Function	2
f_{60}	W / Wavy Function	n
f_{61}	Weierstrass	n
f_{62}	Whitley	n
f_{63}	Xin-She Yang's Function No.01	n
f_{64}	Xin-She Yang's Function No.02	n
f_{65}	Xin-She Yang's Function No.03	n
f_{66}	Xin-She Yang's Function No.06	n

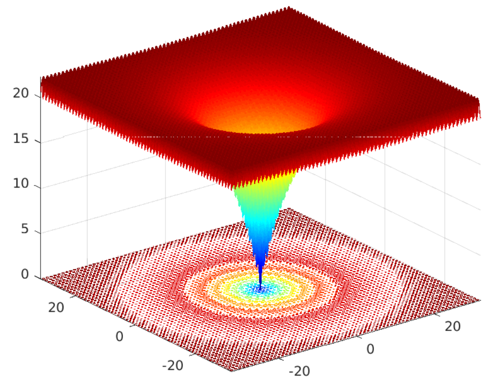
Table A.2: Base optimization functions - part 2

	Function name	**Dim**	**Composed of**
f_{67}	Expanded Griewanks plus Rosenbrocks Function	n	f_{22}, f_{42}
f_{68}	Expanded Scaffers F6 Function	n	f_{45}
f_{69}	Stacked Bird, Egg-crate, Leon, Sawtoothxy	8	f_7, f_{19}, f_{28}, f_{44}
f_{70}	Stacked Adjiman, Cross in Tray, Crowned cross, Schaffer F6	8	f_2, f_{12}, f_{13}, f_{45}
f_{71}	Stacked Davis, Downhill step, Drop-wave, Six-Hump Camel-back	8	f_{14}, f_{17}, f_{18}, f_{52}
f_{72}	Stacked Giunta, Hosaki, Mishra F3, Ursem F3	8	f_{21}, f_{26}, f_{31}, f_{57}

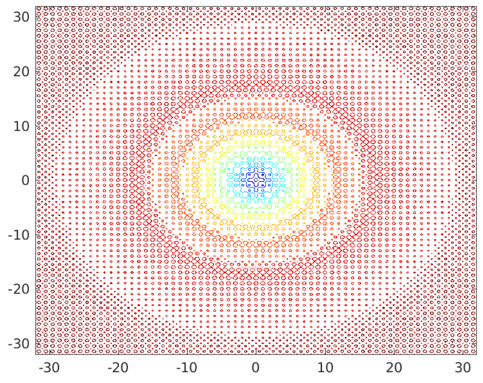
Table A.3: Compound optimization functions

A.3 Basis functions

A.3.1 Ackley's Function



(a) Plot



(b) Contours

Figure A.1: Ackley's function

$$f_1(\mathbf{x}) = -20\exp\left(-0.2\sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}\right) - \exp\left(\frac{1}{n}\sum_{i=1}^{n}\cos(2\pi x_i)\right) + 20 + e \tag{A.1}$$

Bounds

$$-32 \leq x_i \leq 32$$

Optima

$$f_{1,min}(\mathbf{x}^*) = 0$$
$$x_i^* = 0$$

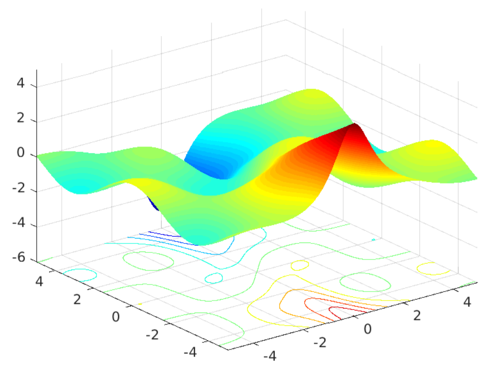
Properties

- n-dimensional
- Multimodal
- Non-separable

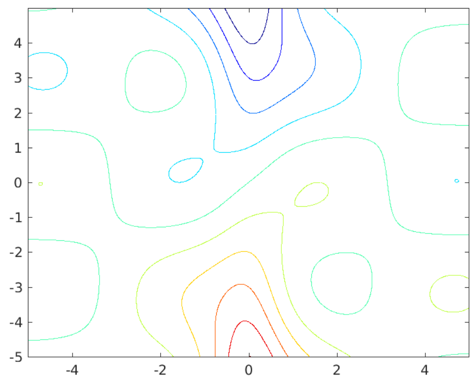
Resources

- `https://www.sfu.ca/~ssurjano/ackley.html`
- `http://infinity77.net/global_optimization/test_functions_nd_A.html`

A.3.2 Adjiman's Function



(a) Plot



(b) Contours

Figure A.2: Adjiman's Function

$$f_2(\mathbf{x}) = \cos(x_1)\sin(x_2) - \frac{x_1}{x_2^2+1} \tag{A.2}$$

Bounds

$$-5 \leq x_i \leq 5$$

Optima

$$f_{2,min}(\mathbf{x}^*) = -x_1^{max}$$
$$x_i^* = (x_1^{max}, 0) = (5, 0)$$

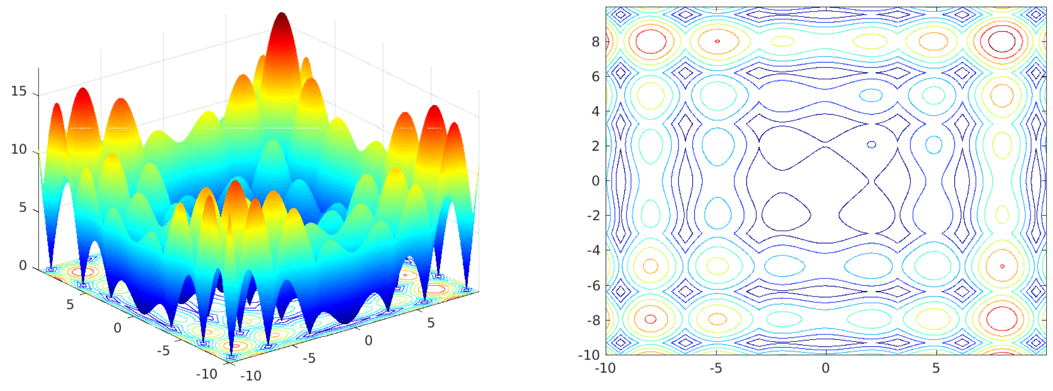
Properties

- 2-dimensional
- Non-Separable
- Multimodal

Resources

- `http://al-roomi.org/benchmarks/unconstrained/2-dimensions/113-adjiman-s-function`
- `http://infinity77.net/global_optimization/test_functions_nd_A.html`

A.3.3 Alpine Function No.01



(a) Plot (b) Contours

Figure A.3: Alpine Function No.01

$$f_3(\mathbf{x}) = \sum_{i=1}^{N} |x_i \sin(x_i) + 0.1x_i| \tag{A.3}$$

Bounds

$$-10 \leq x_i \leq 10$$

Optima

$$f_{3,min}(\mathbf{x}^*) = 0$$
$$x_i^* = 0$$

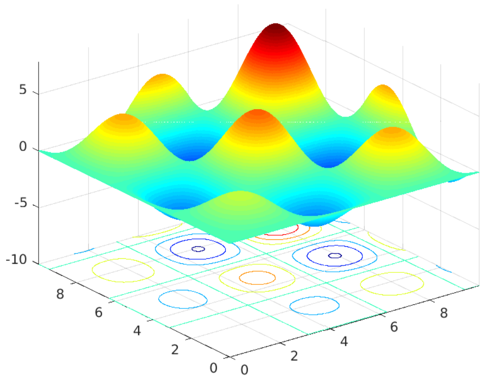
Properties

- Separable
- Multimodal

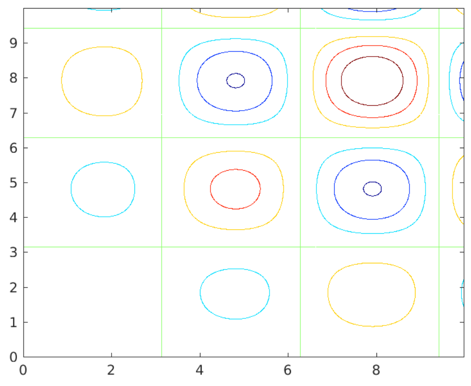
Resources

- `http://al-roomi.org/benchmarks/unconstrained/n-dimensions/162-alpine-function-no-1`
- `http://infinity77.net/global_optimization/test_functions_nd_A.html`

A.3.4 Alpine Function No.02



(a) Plot



(b) Contours

Figure A.4: Alpine Function No.02

$$f_4(\mathbf{x}) = \prod_{i=1}^{n} \sin(x_i) \cdot \sqrt{x_i} \tag{A.4}$$

Bounds

$$0 \leq x_i \leq 10$$

Optima

$$f_{4,min}(\mathbf{x}^*) = 2.808131180007005329 1^N$$

$$x_i^* = 7.917052698245946 2172$$

Properties

- Separable
- Multimodal

Resources

- `http://al-roomi.org/benchmarks/unconstrained/n-dimensions/163-alpine-function-no-2`
- `http://infinity77.net/global_optimization/test_functions_nd_A.html`

A.3.5 ANNs XOR Function

$$f_5(\mathbf{x}) = g_1(\mathbf{x}) + g_2(\mathbf{x}) + g_3(\mathbf{x}) + g_4(\mathbf{x}) \tag{A.5}$$

where:

$$g_1(\mathbf{x}) = \left\{1 + e^{\left[-\frac{x_7}{1+e^{-(x_1+x_2+x_5)}} - \frac{x_8}{1+e^{-(x_3+x_4+x_6)}} - x_9\right]}\right\}^{-2}$$

$$g_2(\mathbf{x}) = \left\{1 + e^{\left[-\frac{x_7}{1+e^{-x_5}} - \frac{x_8}{1+e^{-x_6}} - x_9\right]}\right\}^{-2}$$

$$g_3(\mathbf{x}) = \left\{1 - \frac{1}{1 + e^{\left[-\frac{x_7}{1+e^{-(x_1+x_5)}} - \frac{x_8}{1+e^{-(x_3+x_6)}} - x_9\right]}}\right\}^{2}$$

$$g_4(\mathbf{x}) = \left\{1 - \frac{1}{1 + e^{\left[-\frac{x_7}{1+e^{-(x_2+x_5)}} - \frac{x_8}{1+e^{-(x_4+x_6)}} - x_9\right]}}\right\}^{2}$$

Bounds

$$-1 \leq x_i \leq 1$$

Optima

$$f_{5,min}(\mathbf{x}^*) \approx 0.959759$$

$x_i^* \approx (0.99999, 0.99993, -0.89414, 0.99994, 0.55932, 0.99994, 0.99994, -0.99963, -0.08272)$

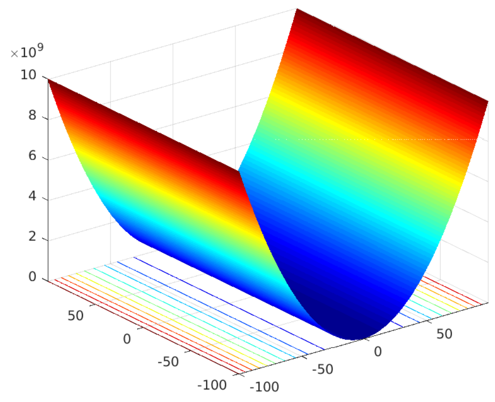
Properties

- 9-dimensional
- Multimodal
- Non-Separable

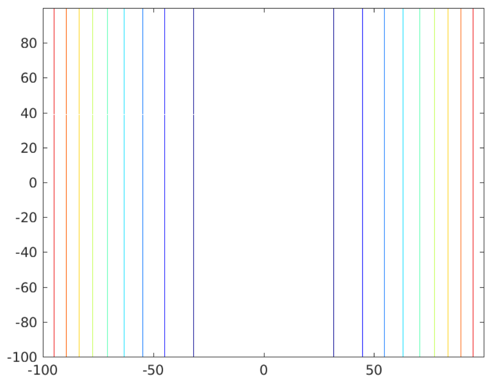
Resources

- `http://al-roomi.org/benchmarks/unconstrained/9-dimensions/145-anns-xor-function`

A.3.6 Bent cigar



(a) Plot



(b) Contours

Figure A.5: Bent cigar Function

$$f_6(\mathbf{x}) = x_1^2 + 10^6 \sum_{i=2}^{n} x_i^2 \tag{A.6}$$

Bounds

$$-100 \leq x_i \leq 100$$

Optima

$$f_{6,min}(\mathbf{x}^*) = 0$$
$$x_i^* = 0$$

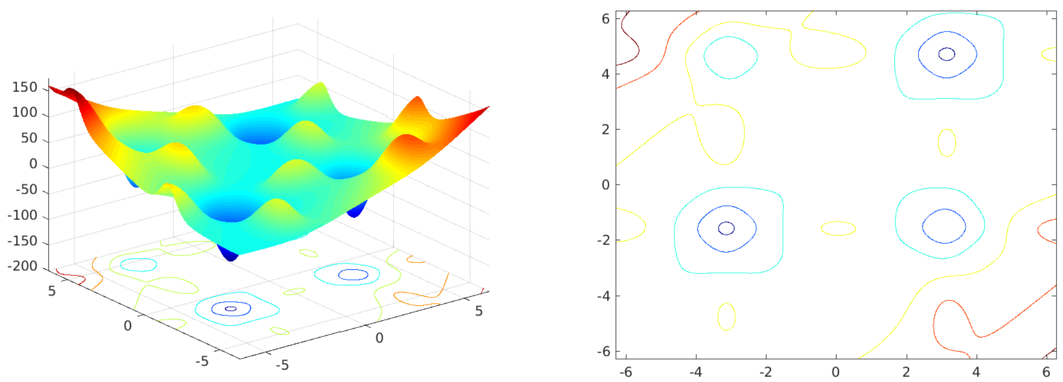
Properties

- n-dimensional
- Unimodal

Resources

- `http://al-roomi.org/benchmarks/unconstrained/n-dimensions/164-bent-cigar-function`

A.3.7 Bird Function



(a) Plot (b) Contours

Figure A.6: Bird Function

$$f_7(\mathbf{x}) = (x_1 - x_2)^2 + \sin(x_1) \cdot e^{[1-\cos(x_2)]^2} + \cos(x_2) \cdot e^{[1-\sin(x_1)]^2} \tag{A.7}$$

Bounds

$$-2\pi \leq x_i \leq 2\pi$$

Optima

$$f_{7,min}(\mathbf{x}^*) = -106.7645367198034$$

$$\mathbf{x}^* \approx (4.701055751981055, 3.152946019601391)\,, \\ (-1.582142172055011, -3.130246799635430)$$

Properties

- 2-dimensional
- Non-Separable
- Multimodal

Resources

- `http://infinity77.net/global_optimization/test_functions_nd_B.html`

A.3.8 Brad Function

$$f_8(\mathbf{x}) = \sum_{i=1}^{15} \left[\frac{y_i - x_1 - u_i}{v_i x_2 + w_i x_3} \right]^2 \tag{A.8}$$

with

$$u_i = i$$

$$v_i = 16 - i$$

$$w_i = \min(u_i, v_i)$$

$$\mathbf{y} = [0.14, 0.18, 0.22, 0.25, 0.29, 0.32, 0.35, 0.39, 0.37, 0.58, 0.73, 0.96, 1.34, 2.10, 4.39]^T$$

Bounds

$$-0.25 \leq x_1 \leq 0.25$$

$$0.01 \leq x_2, x_3 \leq 2.5$$

Optima

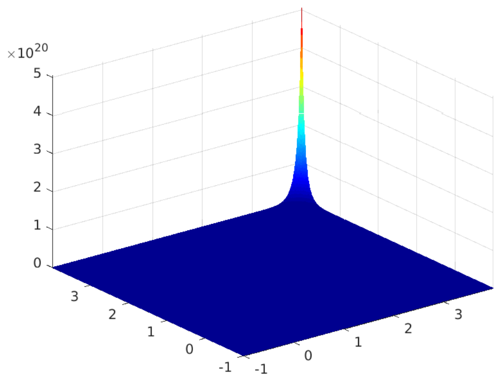
$$f_{8,min}(X^*) = 0.00821487$$

$$\mathbf{x}^* = (0.0824, 1.133, 2.3437)$$

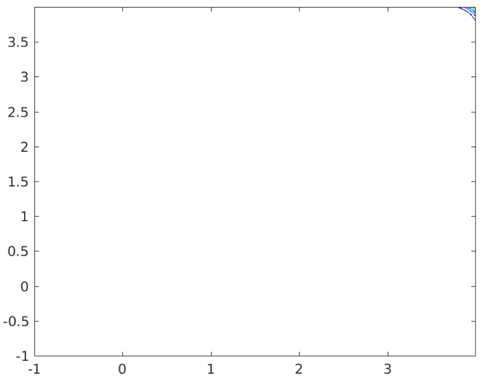
Properties

- 3-dimensional
- Non-Separable
- Multimodal

A.3.9 Brown Function



(a) Plot



(b) Contours

Figure A.7: Brown Function

$$f_9(\mathbf{x}) = \sum_{i=1}^{n-1} \left(x_i^2\right)^{\left(x_{i+1}^2+1\right)} + \left(x_{i+1}^2\right)^{\left(x_i^2+1\right)} \tag{A.9}$$

Bounds

$$-1 \leq x_i \leq 4$$

Optima

$$f_{9,min}(\mathbf{x}^*) = 0$$
$$x_i^* = 0$$

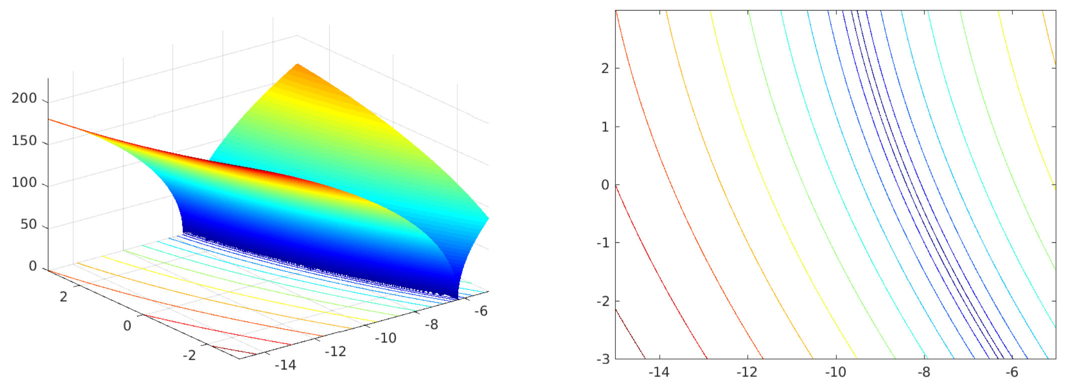
Properties

- n-dimensional
- Non-Separable
- Unimodal

Resources

- `http://al-roomi.org/benchmarks/unconstrained/n-dimensions/241-brown-s-function`

A.3.10 Bukin's Function No.06



(a) Plot (b) Contours

Figure A.8: Bukin's Function No.06

$$f_{10}(\mathbf{x}) = 100\sqrt{\left|x_2 - 0.01x_1^2\right|} + 0.01\left|x_1 + 10\right| \tag{A.10}$$

Bounds

$$-15 \leq x_1 \leq -5, -3 \leq x_2 \leq 3$$

Optima

$$f_{10,min}(\mathbf{x}^*) = 0$$
$$x_i^* = (-10, 1)$$

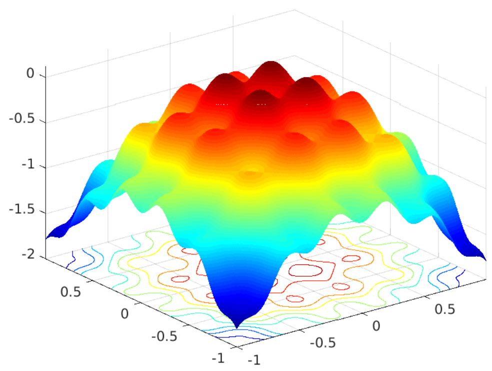
Properties

- 2-dimensional
- Non-Separable
- Multimodal

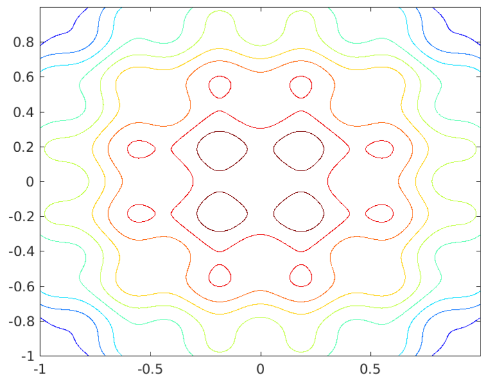
Resources

- `http://al-roomi.org/benchmarks/unconstrained/2-dimensions/52-bukin-s-function-no-`

A.3.11 Cosine Mixture Function



(a) Plot



(b) Contours

Figure A.9: Cosine Mixture Function

$$f_{11}(\mathbf{x}) = 0.1 \sum_{i=1}^{n} \cos\left(5\pi x_i\right) - \sum_{i=1}^{n} x_i^2 \tag{A.11}$$

Bounds

$$-1 \leq x_i \leq 1$$

Optima

$$f_{11,max}(\mathbf{x}^*) = 0.1 \text{ for } n = 1 \text{ , } f_{11,max}(\mathbf{x}^*) = 0.2 \text{ for } n = 2 \text{ etc...}$$

$$x_i^* = 0$$

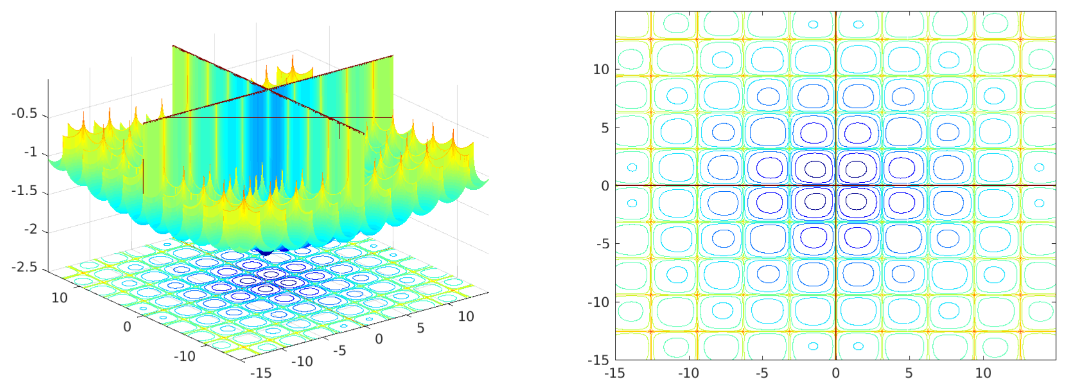
Properties

- n-dimensional
- Separable
- Multimodal

Resources

- `http://infinity77.net/global_optimization/test_functions_nd_C.html`
- `http://al-roomi.org/benchmarks/unconstrained/n-dimensions/166-cosine-mixture-function`

A.3.12 Cross-In Tray Function



(a) Plot (b) Contours

Figure A.10: Cross-In Tray Function

$$f_{12}(\mathbf{x}) = -0.0001\left(\left|\sin(x_1)\sin(x_2)\, e^{\left|100-\frac{\sqrt{x_1^2+x_2^2}}{\pi}\right|}\right|+1\right)^{0.1} \tag{A.12}$$

Bounds

$$-15 \leq x_i \leq 15$$

Optima

$$f_{12,min}(\mathbf{x}^*) = -2.062611870822739$$

$$x_i^* = \pm 1.349406608602084$$

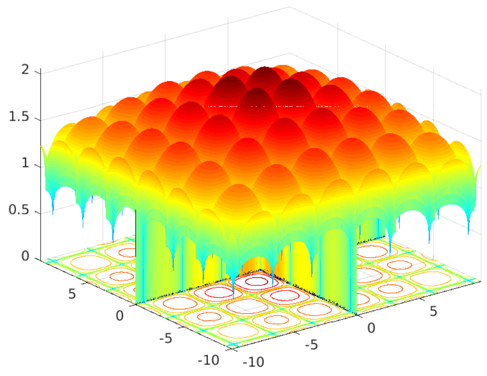
Properties

- 2-dimensional
- Non-Separable
- Multimodal

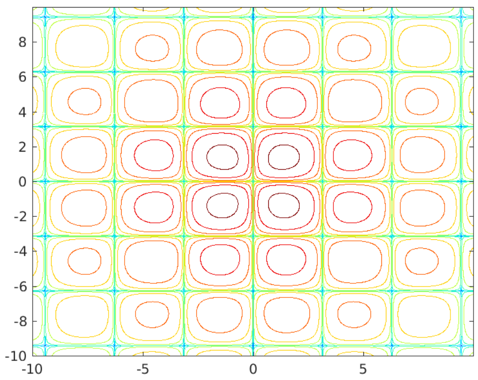
Resources

- `http://infinity77.net/global_optimization/test_functions_nd_C.html`
- `http://al-roomi.org/benchmarks/unconstrained/2-dimensions/44-cross-in-tray-functi`

A.3.13 Crowned Cross Function



(a) Plot



(b) Contours

Figure A.11: Crowned Cross Function

$$f_{13}(\mathbf{x}) = 0.0001\left(\left|\sin(x_1)\sin(x_2)\,e^{\left|100-\frac{\sqrt{x_1^2+x_2^2}}{\pi}\right|}\right|+1\right)^{0.1} \tag{A.13}$$

Bounds

$$-10 \leq x_i \leq 10$$

Optima

$$f_{13,min}(\mathbf{x}^*) = 0.0001$$
$$x_i^* = 0$$

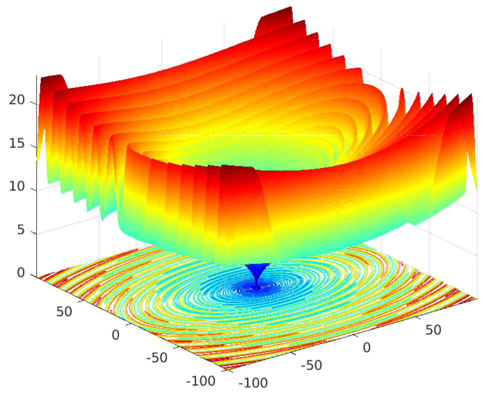
Properties

- 2-dimensional
- Non-Separable
- Multimodal

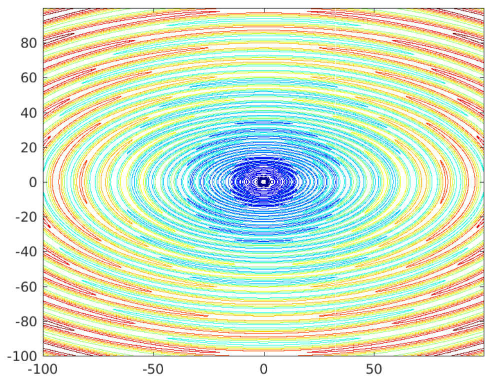
Resources

- `http://al-roomi.org/benchmarks/unconstrained/2-dimensions/46-crowned-cross-function`

A.3.14 Davis' Function



(a) Plot



(b) Contours

Figure A.12: Davis' Function

$$f_{14}(\mathbf{x}) = \left(x_1^2 + x_2^2\right)^{0.25} \left[\sin^2\left(50\left(3x_1^2 + x_2^2\right)^{0.1}\right) + 1\right] \tag{A.14}$$

Bounds

$$-100 \leq x_i \leq 100$$

Optima

$$f_{min}(\mathbf{x}^*) = 0$$
$$x_i^* = 0$$

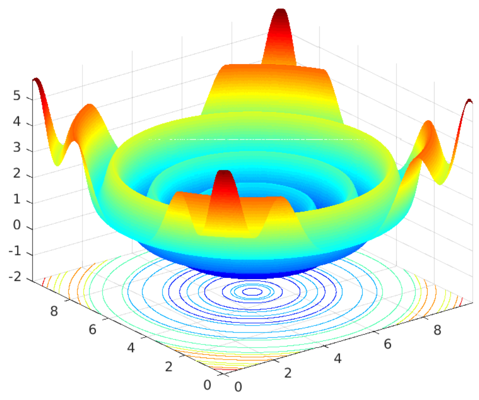
Properties

- 2-dimensional
- Multimodal

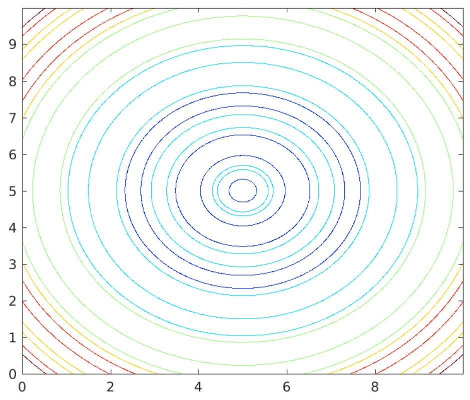
Resources

- `http://al-roomi.org/benchmarks/unconstrained/2-dimensions/47-davis-function`

A.3.15 Deflected Corrugated Spring Function



(a) Plot



(b) Contours

Figure A.13: Deflected Corrugated Spring Function

$$f_{15}(\mathbf{x}) = 0.1 \times \sum_{i=1}^{n} (x_i - \alpha)^2 - \cos\left(k \times \sqrt{\sum_{i=1}^{n} (x_i - \alpha)^2}\right) \tag{A.15}$$

with

$$a = k = 5$$

Bounds

$$0 \leq x_i \leq 2\alpha$$

Optima

$$f_{15,min}(\mathbf{x}^*) = -1$$
$$x_i^* = \alpha$$

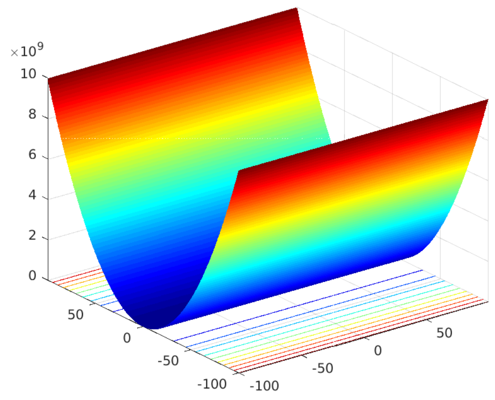
Properties

- n-dimensional
- Multimodal

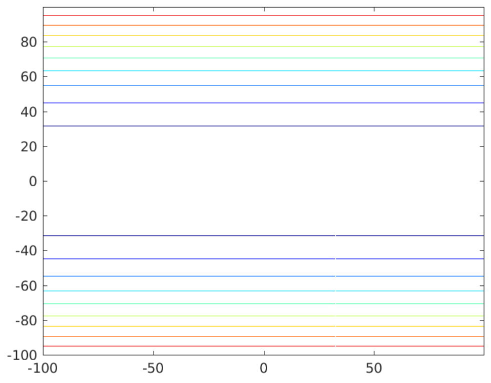
Resources

- `http://al-roomi.org/benchmarks/unconstrained/n-dimensions/238-deflected-corrugated-spri`
- `http://infinity77.net/global_optimization/test_functions_nd_D.html`

A.3.16 Discus Function



(a) Plot



(b) Contours

Figure A.14: Discuss function

$$f_{16}(\mathbf{x}) = 10^6 x_1^2 + \sum_{i=1}^{n} x_i^2 \tag{A.16}$$

Bounds

$$-100 \leq x_i \leq 100$$

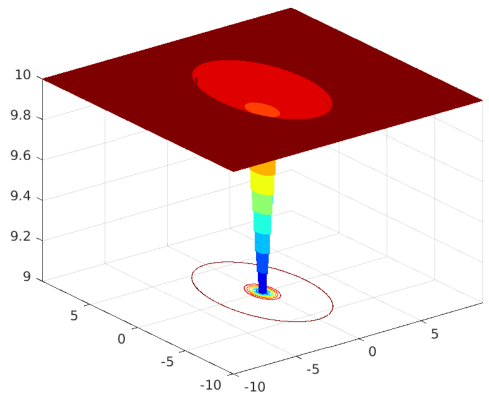
Optima

$$f_{16,min}(\mathbf{x}^*) = 0$$
$$x_i^* = 0$$

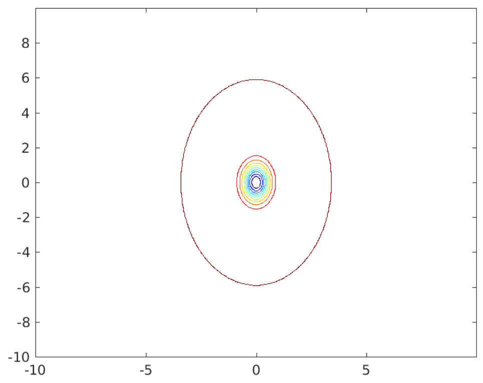
Properties

- n-dimensional
- Separable
- Unimodal

A.3.17 Downhill Step Function



(a) Plot



(b) Contours

Figure A.15: Downhill Step Function

$$f_{17}(\mathbf{x}) = \frac{\left\lfloor 10 \times \left(10 - e^{-x_1^2 - 3x_2^2}\right) \right\rfloor}{10} \tag{A.17}$$

Bounds

$$-10 \leq x_i \leq 10$$

Optima

$$f_{17,min}(\mathbf{x}^*) = 9$$
$$x_i^* = (0, 0)$$

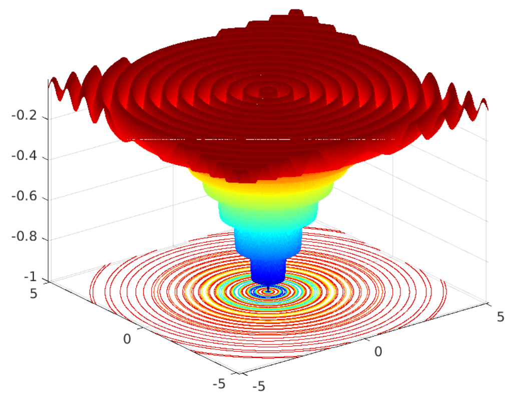
Properties

- 2-dimensional
- Separable
- Unimodal

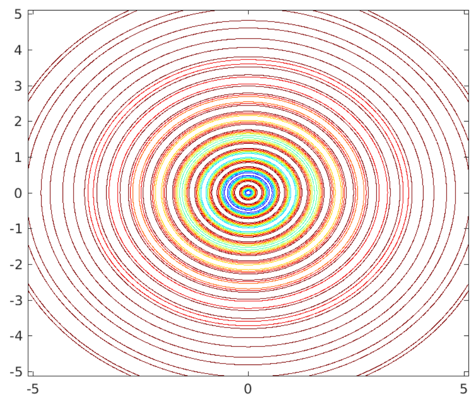
Resources

- `http://al-roomi.org/benchmarks/unconstrained/2-dimensions/114-downhill-step-function`

A.3.18 Drop-Wave Function



(a) Plot



(b) Contours

Figure A.16: Drop-Wave Function

$$f_{18}(\mathbf{x}) = -\frac{1+\cos\left(12\sqrt{x_1^2+x_2^2}\right)}{\frac{1}{2}\left(x_1^2+x_2^2\right)+2} \tag{A.18}$$

Bounds

$$-5.12 \leq x_i \leq 5.12$$

Optima

$$f_{18,min}(\mathbf{x}^*) = -1$$
$$x_i^* = 0$$

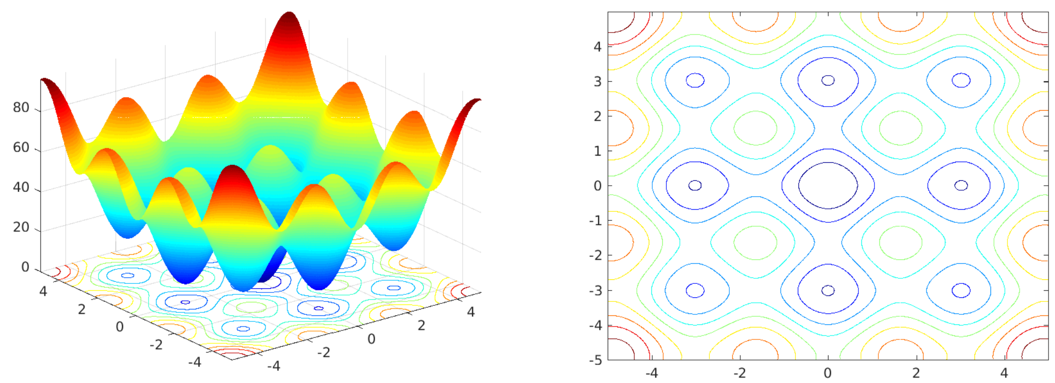
Properties

- 2-dimensional
- Non-separable
- Multimodal

Resources

- `https://www.sfu.ca/~ssurjano/drop.html`
- `http://al-roomi.org/benchmarks/unconstrained/2-dimensions/54-drop-wave-function`
- `http://infinity77.net/global_optimization/test_functions_nd_D.html`

A.3.19 Egg Crate Function



(a) Plot (b) Contours

Figure A.17: Egg Crate Function

$$f_{19}(\mathbf{x}) = x_1^2 + x_2^2 + 25\left[\sin^2(x_1) + \sin^2(x_2)\right] \tag{A.19}$$

Bounds

$$-5 \leq x_i \leq 5$$

Optima

$$f_{19,min}(\mathbf{x}^*) = 0$$
$$x_i^* = 0$$

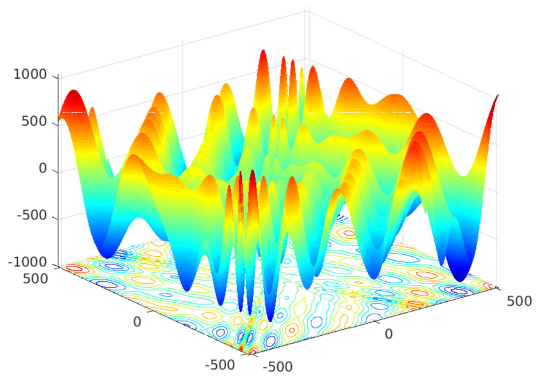
Properties

- 2-dimensional
- Separable

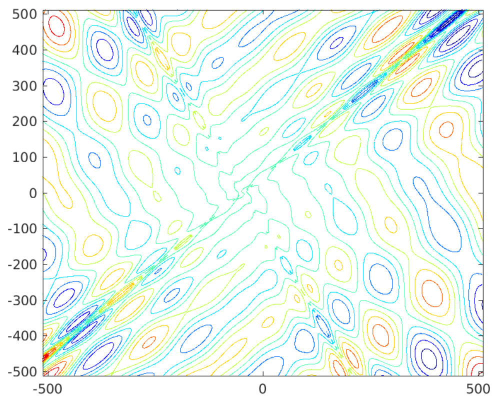
Resources

- `http://benchmarkfcns.xyz/benchmarkfcns/eggcratefcn.html`
- `http://al-roomi.org/benchmarks/unconstrained/2-dimensions/122-egg-crate-function`
- `http://infinity77.net/global_optimization/test_functions_nd_E.html`

A.3.20 Egg-Holder Function



(a) Plot



(b) Contours

Figure A.18: Egg-Holder Function

$$f_{20}(\mathbf{x}) = \sum_{i=1}^{n-1} \left[-x_i \sin\left(\sqrt{|x_i - x_{i+1} - 47|}\right) - (x_{i+1} + 47) \sin\left(\sqrt{|0.5x_i + x_{i+1} + 47|}\right) \right] \tag{A.20}$$

Bounds

$$512 \leq x_i \leq 512$$

Optima

$$f_{20,min}(\mathbf{x}^*) = -959.640662720850742 \text{ for } n = 2$$
$$\mathbf{x}^* = (512, 404.231805123817)$$

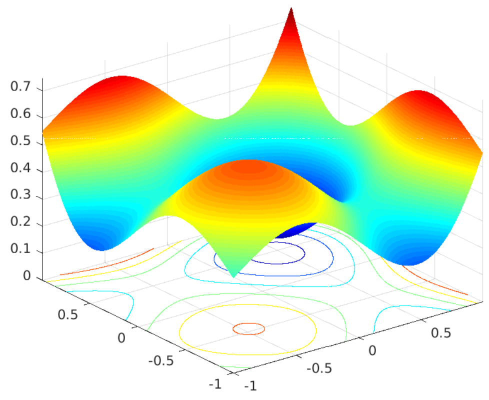
Properties

- n-dimensional
- Non-Separable
- Multimodal

Resources

- `https://www.sfu.ca/~ssurjano/egg.html`
- `http://al-roomi.org/benchmarks/unconstrained/n-dimensions/187-egg-holder-function`
- `http://infinity77.net/global_optimization/test_functions_nd_E.html`

A.3.21 Giunta's Function


(a) Plot

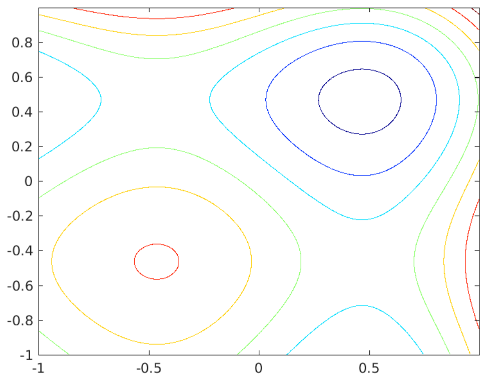

(b) Contours

Figure A.19: Giunta's Function

$$f_{21}(\mathbf{x}) = 0.6 + \sum_{i=1}^{n} \left[\sin^2\left(1 - \frac{16}{15}x_i\right) - \frac{1}{50}\sin\left(4 - \frac{64}{15}x_i\right) - \sin\left(1 - \frac{16}{15}x_i\right)\right] \tag{A.21}$$

Bounds

$$-1 \leq x_i \leq 1$$

Optima

$$f_{21,min}(\mathbf{x}^*) = 0.06447042053690566$$
$$\mathbf{x}^* = (0.4673200277395354, 0.4673200169591304)$$

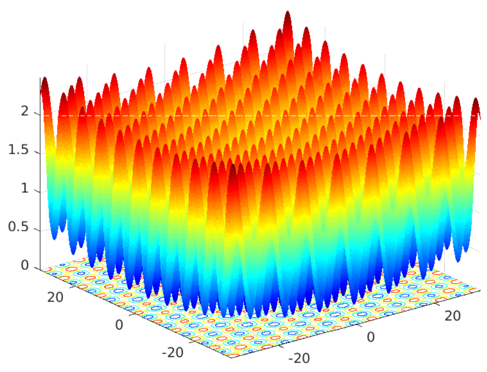
Properties

- N-dimensional
- Separable
- Multimodal

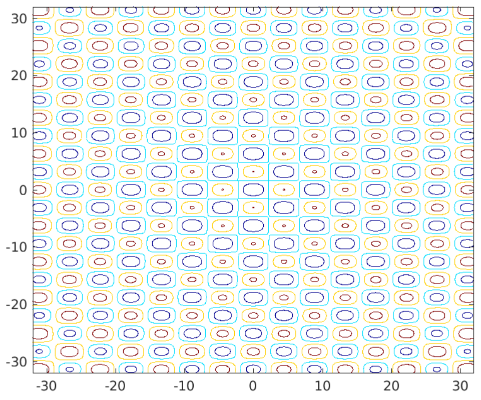
Resources

- `http://al-roomi.org/benchmarks/unconstrained/2-dimensions/154-giunta-s-function`
- `http://infinity77.net/global_optimization/test_functions_nd_G.html`

A.3.22 Griewank



(a) Plot



(b) Contours

Figure A.20: Griewank function

$$f_{22}(\mathbf{x}) = \frac{1}{4000}\sum_{i=1}^{n} x_i^2 - \prod_{i=1}^{n}\cos\left(\frac{x_i}{\sqrt{i}}\right) + 1 \qquad \text{(A.22)}$$

Bounds

$$-600 \le x_i \le 600$$

Optima

$$f_{22,min}(\mathbf{x}^*) = 0$$
$$x_i^* = 0$$

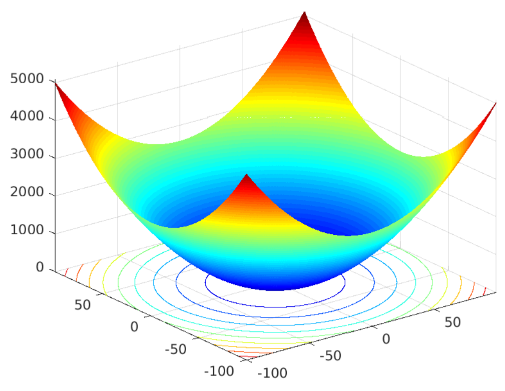
Properties

- n-dimensional
- Non-Separable
- Multimodal

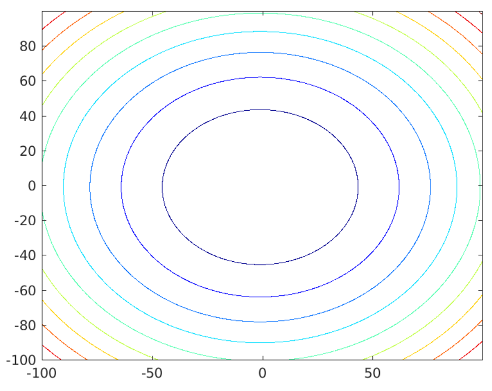
Resources

- `http://infinity77.net/global_optimization/test_functions_nd_G.html`
- `https://www.sfu.ca/~ssurjano/griewank.html`
- `http://mathworld.wolfram.com/GriewankFunction.html`

A.3.23 HappyCat Function



(a) Plot



(b) Contours

Figure A.21: HappyCat function

$$f_{23}(\mathbf{x}) = \left| \sum_{i=1}^{n} x_i^2 - N \right|^{1/4} + \frac{(0.5 \sum_{i=1}^{n} x_i^2 + \sum_{i=1}^{n} x_i)}{N} + 0.5 \tag{A.23}$$

Bounds

$$-100 \leq x_i \leq 100$$

Optima

Unknown

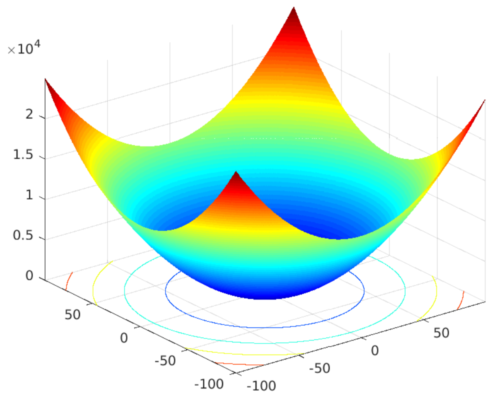
Properties

- n-dimensional
- Non-separable
- Multimodal

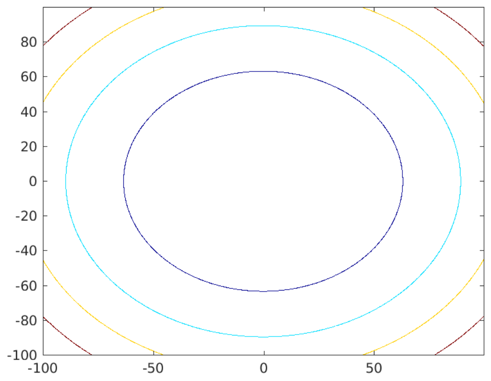
Resources

- J. Liang, B. Qu, and P. Suganthan, "Problem definitions and evaluation criteria for the cec 2014 special session and competition on single objective real-parameter numerical optimization," *Computational Intelligence Laboratory, Zhengzhou University, Zhengzhou China and Technical Report, Nanyang Technological University, Singapore*, 2013

A.3.24 HGBat Function



(a) Plot



(b) Contours

Figure A.22: HGBat function

$$f_{24}(\mathbf{x}) = \left| \left(\sum_{i=1}^{n} x_i^2 \right)^2 - \left(\sum_{i=1}^{n} x_i \right)^2 \right|^{1/4} + \frac{(0.5 \sum_{i=1}^{n} x_i^2 + \sum_{i=1}^{n} x_i)}{N} + 0.5 \qquad (A.24)$$

Bounds

$$-100 \leq x_i \leq 100$$

Optima

Unknown

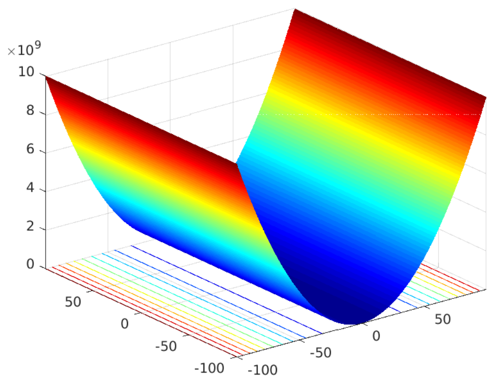
Properties

- n-dimensional
- Non-separable
- Multimodal

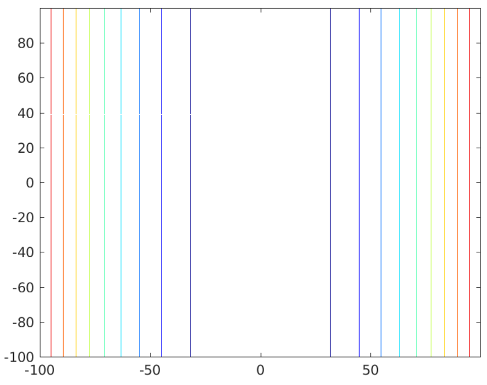
Resources

- J. Liang, B. Qu, and P. Suganthan, "Problem definitions and evaluation criteria for the cec 2014 special session and competition on single objective real-parameter numerical optimization," *Computational Intelligence Laboratory, Zhengzhou University, Zhengzhou China and Technical Report, Nanyang Technological University, Singapore*, 2013

A.3.25 High Conditioned Elliptic Function



(a) Plot



(b) Contours

Figure A.23: High Conditioned Elliptic Function

$$f_{25}(\mathbf{x}) = \sum_{i=1}^{n} (10^6)^{\frac{i-1}{N-1}} x_i^2 \tag{A.25}$$

Bounds

$$-100 \leq x_i \leq 100$$

Optima

Unknown

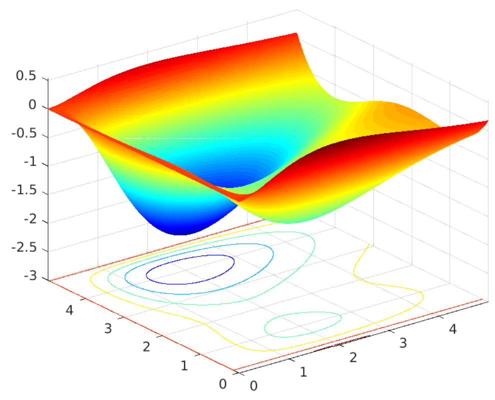
Properties

- n-dimensional
- Separable
- Unimodal

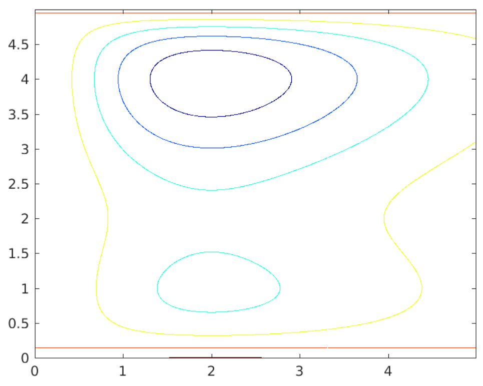
Resources

- J. Liang, B. Qu, and P. Suganthan, "Problem definitions and evaluation criteria for the cec 2014 special session and competition on single objective real-parameter numerical optimization," *Computational Intelligence Laboratory, Zhengzhou University, Zhengzhou China and Technical Report, Nanyang Technological University, Singapore*, 2013

A.3.26 Hosaki's Function



(a) Plot



(b) Contours

Figure A.24: Hosaki's Function

$$f_{26}(\mathbf{x}) = \left(1 - 8x_1 + 7x_1^2 - \frac{7}{3}x_1^3 + \frac{1}{4}x_1^4\right) x_2^2 e^{-x_2} \tag{A.26}$$

Bounds

$$0 \leq x_i \leq 10$$

Optima

$$f_{26,min}(\mathbf{x}^*) = -2.345811576101292$$

$$\mathbf{x}^* = (4, 2)$$

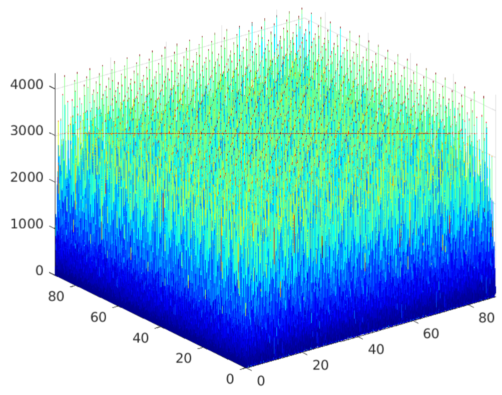
Properties

- 2-dimensional
- Non-Separable
- Multimodal

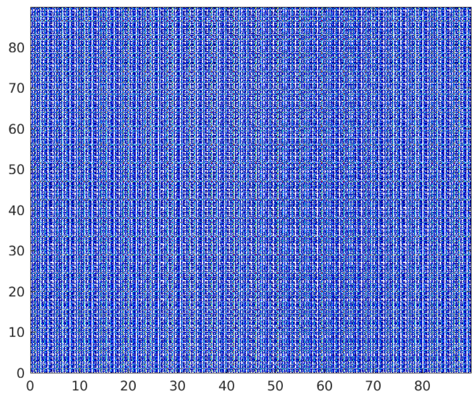
Resources

- `http://al-roomi.org/benchmarks/unconstrained/2-dimensions/58-hosaki-s-function`
- `http://infinity77.net/global_optimization/test_functions_nd_H.html`

A.3.27 Katsuura Function



(a) Plot



(b) Contours

Figure A.25: Katsuura Function function

$$f_{27}(\mathbf{x}) = \frac{10}{n^2}\prod_{i=1}^{n}(1 + i\sum_{j=1}^{32}\frac{|2^j x_i - \text{round}(2^j x_i)|}{2^j})^{\frac{10}{n^{1.2}}} - \frac{10}{n^2} \tag{A.27}$$

Bounds

$$-100 \leq x_i \leq 100$$

Optima

$$f_{27,min}(\mathbf{x}^*) = 0$$
$$x_i^* = 0$$

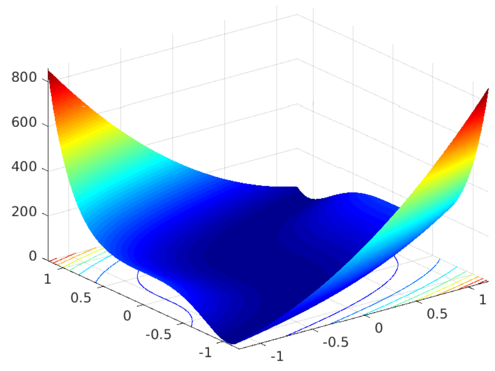
Properties

- n-dimensional
- Non-Separable
- Multimodal

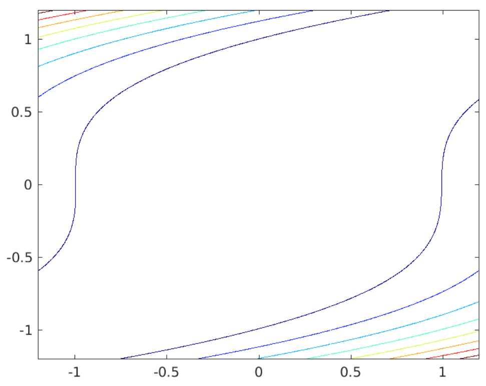
Resources

- J. Liang, B. Qu, and P. Suganthan, "Problem definitions and evaluation criteria for the cec 2014 special session and competition on single objective real-parameter numerical optimization," *Computational Intelligence Laboratory, Zhengzhou University, Zhengzhou China and Technical Report, Nanyang Technological University, Singapore*, 2013

A.3.28 Leon's Function



(a) Plot



(b) Contours

Figure A.26: Leon's Function

$$f_{28}(\mathbf{x}) = 100\left(x_2 - x_1^3\right)^2 + (1 - x_1)^2 \tag{A.28}$$

Bounds

$$-1.2 \leq x_i \leq 1.2$$

Optima

$$f_{28,min}(\mathbf{x}^*) = 0$$
$$\mathbf{x}^* = (1, 1)$$

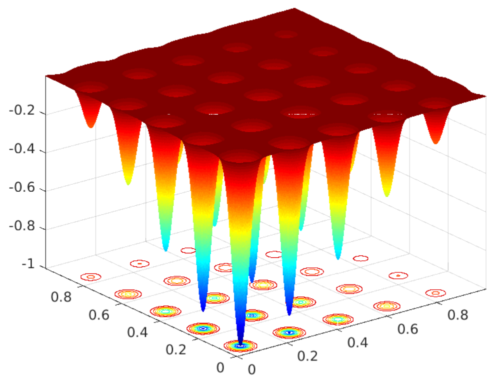
Properties

- 2-dimensional
- Non-Separable
- Unimodal

Resources

- `http://al-roomi.org/benchmarks/unconstrained/2-dimensions/125-leon-s-function`

A.3.29 L (or F2) Function


(a) Plot

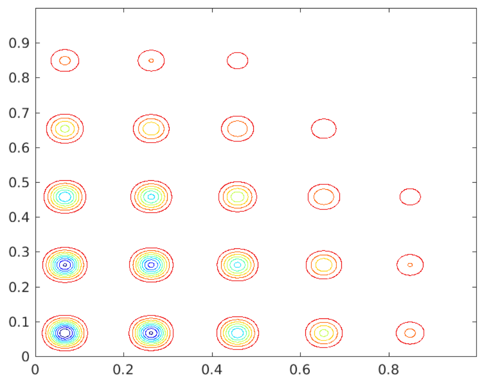

(b) Contours

Figure A.27: L (or F2) Function

$$f_{29}(\mathbf{x}) = -\prod_{i=1}^{n} \sin^k \left(l_1 \pi x_i + l_2\right) \cdot e^{-l_3\left(\frac{x_i - l_4}{l_5}\right)^2} \tag{A.29}$$

With:

$$k = 6, l_1 = 5.1, l_2 = 0.5, l_3 = 4\ln(2), l_4 = 0.066832364099628, l_5 = 0.64$$

Bounds

$$0 \leq x_i \leq 1$$

Optima

$$f_{29,min}(\mathbf{x}^*) = -1$$
$$x_i^* = l_4$$

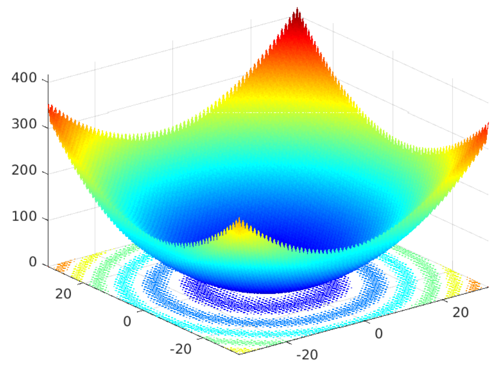
Properties

- n-dimensional
- Non-seperable
- Multimodal

Resources

- `http://al-roomi.org/benchmarks/unconstrained/n-dimensions/276-l-or-f2-function`

A.3.30 Lunacek's bi-Rastrigin Function


(a) Plot

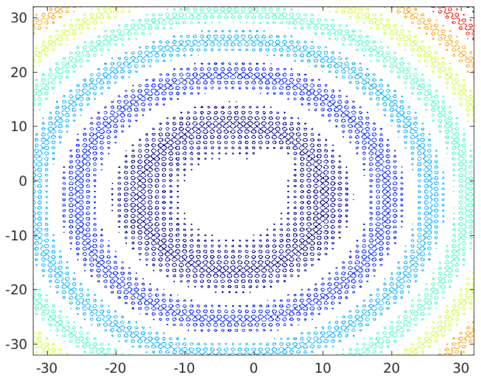

(b) Contours

Figure A.28: Lunacek's bi-Rastrigin Function

$$f_{30}(\mathbf{x}) = \min\left[\sum_{i=1}^{n}(x_i-\mu_1)^2\ ,\ d\cdot n + s\cdot\sum_{i=1}^{n}(x_i-\mu_2)^2\right] + 10\sum_{i=1}^{n}\left\{1-\cos\left[2\pi\left(x_i-\mu_1\right)\right]\right\} \tag{A.30}$$

$$\mu_1 = 2.5$$

$$\mu_2 = -\sqrt{\frac{\mu_1^2-d}{s}}$$

$$d \in 1,2,3,4 \text{ standardized with } d = 1$$

$$s \in [0.2, 1.4] \text{ standardized with } s = 1 - \frac{1}{2\sqrt{2+20}-8.2}$$

Bounds

$$-5.12 \le x_i \le 5.12$$

Optima

For $N = 1, 2$:

$$f_{30,min}(\mathbf{x}^*) = 0$$

$$x_i^* = \mu_1$$

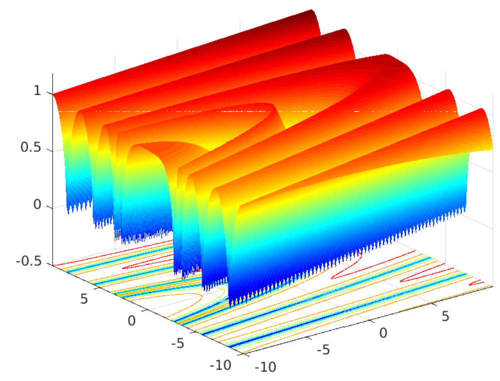
Properties

- n-dimensional
- Multimodal

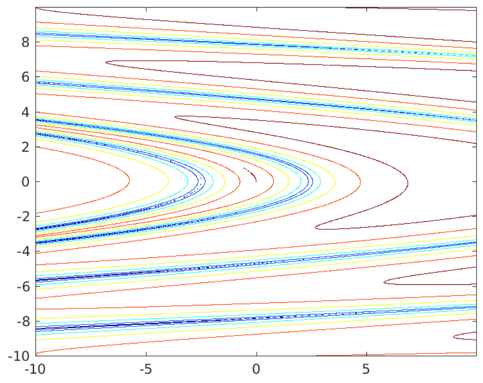
Resources

- `http://al-roomi.org/benchmarks/unconstrained/n-dimensions/229-lunacek-s-bi-rastri`

A.3.31 Mishra's Function No.03



(a) Plot



(b) Contours

Figure A.29: Mishra's Function No.03

$$f_{32}(\mathbf{x}) = \left|\cos\left(\sqrt{|x_1^2 + x_2|}\right)\right|^{0.5} + \frac{x_1 + x_2}{100} \tag{A.31}$$

Bounds

$$-10 \leq x_i \leq 10$$

Optima

$$f_{31,min}(\mathbf{x}^*) = -0.184651333342989$$
$$\mathbf{x}^* = (-8.466613775046579, -9.998521308999999)$$

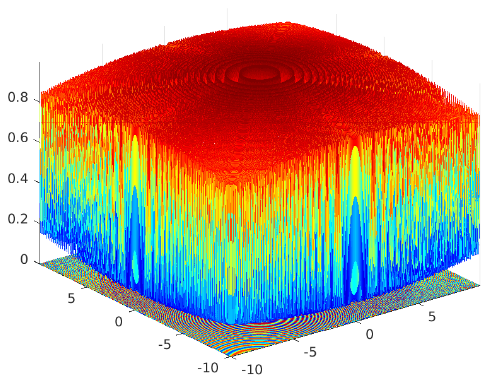
Properties

- 2-dimensional
- Non-Separable
- Multimodal

Resources

- `http://al-roomi.org/benchmarks/unconstrained/2-dimensions/138-mishra-s-function-no-3`

A.3.32 Modified Schaffer's Function No.01


(a) Plot

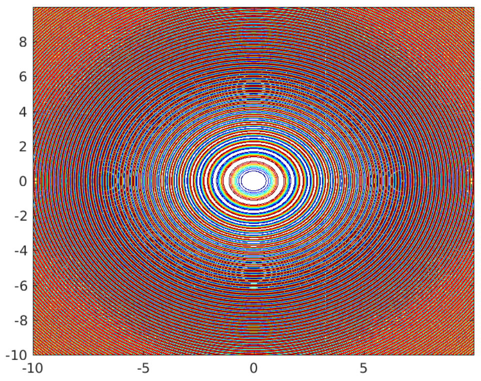

(b) Contours

Figure A.30: Modified Schaffer's Function No.01

$$f_{32}(\mathbf{x}) = 0.5 + \frac{\sin^2\left(x_1^2 + x_2^2\right) - 0.5}{\left[1 + 0.001\left(x_1^2 + x_2^2\right)\right]^2} \tag{A.32}$$

Bounds

$$-100 \leq x_i \leq 100$$

Optima

$$f_{32,min}(\mathbf{x}^*) = 0$$
$$x_i^* = 0$$

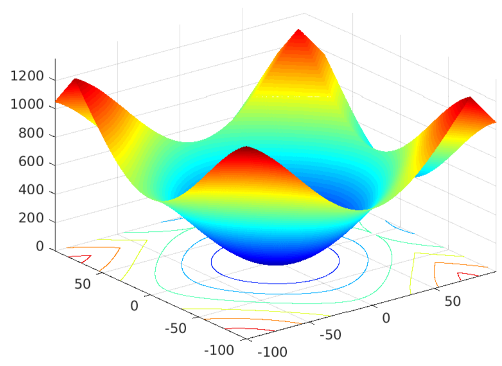
Properties

- 2-dimensional
- Multimodal

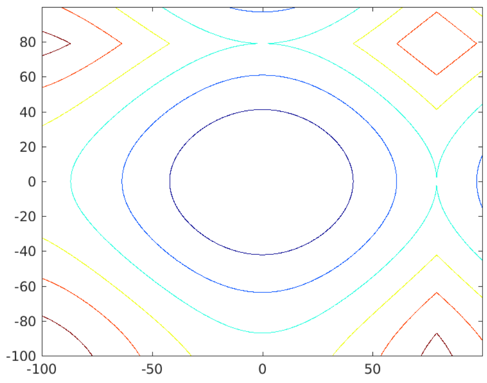
Resources

- `http://al-roomi.org/benchmarks/unconstrained/2-dimensions/96-modified-schaffer-s-`

A.3.33 Modified Schwefel function



(a) Plot



(b) Contours

Figure A.31: Modified Schwefel function

$$f_{33}(\mathbf{x}) = 418.9829n - \sum_{i=1}^{n} g(z_i), \tag{A.33}$$

$$z_i = x_i + 4.209687462275036e + 002$$

$$g(z_i) = \begin{cases} z_i \sin(|z_i|^{1/2}) & \text{if } |z_i| \le 500 \\ (500 - \text{mod}(z_i, 500)) \sin(\sqrt{|500 - \text{mod}(z_i, 500)|}) - \frac{(z_i - 500)^2}{10000n} & \text{if } z_i > 500 \\ (\text{mod}(|z_i|, 500) - 500) \sin(\sqrt{|\text{mod}(|z_i|, 500) - 500|}) - \frac{(z_i + 500)^2}{10000n} & \text{if } z_i < -500 \end{cases}$$

Bounds

$$-100 \le x_i \le 100$$

Optima

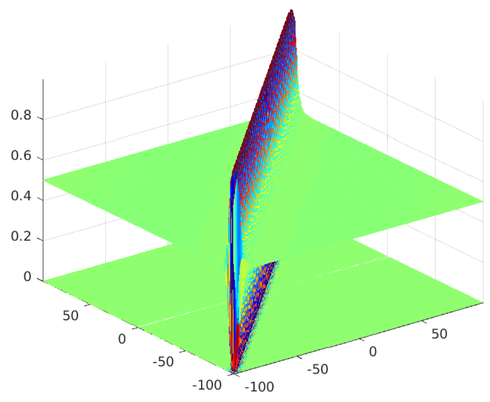
$$f_{33,min}(\mathbf{x}^*) = 0$$

$$x_i^* = 0$$

Properties

- n-dimensional
- Multimodal

A.3.34 Pathological Function


(a) Plot

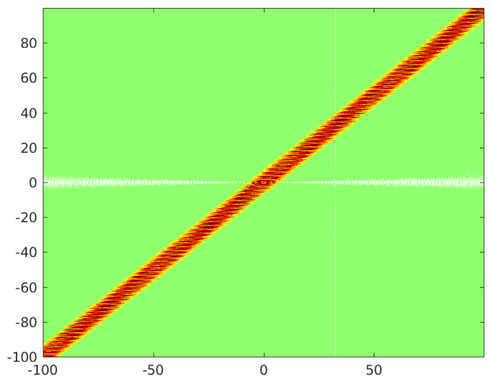

(b) Contours

Figure A.32: Pathological Function

$$f_{34}(\mathbf{x}) = \sum_{i=1}^{n-1} \left[0.5 + \frac{\sin^2\left(\sqrt{100x_i^2 + x_{i+1}^2}\right) - 0.5}{1 + 0.001\left(x_i^2 - 2x_i x_{i+1} + x_{i+1}^2\right)^2} \right] \tag{A.34}$$

Bounds

$$-100 \leq x_i \leq 100$$

Optima

$$f_{34,min}(\mathbf{x}^*) = 0$$
$$x_i^* = 0$$

Properties

- n-dimensional
- Non-Separable
- Multimodal

Resources

- `http://al-roomi.org/benchmarks/unconstrained/n-dimensions/239-pathological-functio`

A.3.35 Paviani

$$f_{35}(\mathbf{x}) = \sum_{i=1}^{10} \left[\log^2(10 - x_i) + \log^2(x_i - 2)\right] - \left(\prod_{i=1}^{10} x_i^{10}\right)^{0.2} \tag{A.35}$$

Bounds

$$2.001 \leq x_i \leq 9.999$$

Optima

$$f_{35,min}(\mathbf{x}^*) = -45.7784684040686$$
$$x_i^* = 9.350266$$

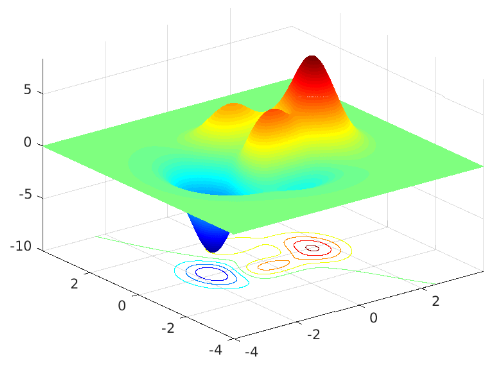
Properties

- 10-dimensional
- Non-Separable
- Multimodal

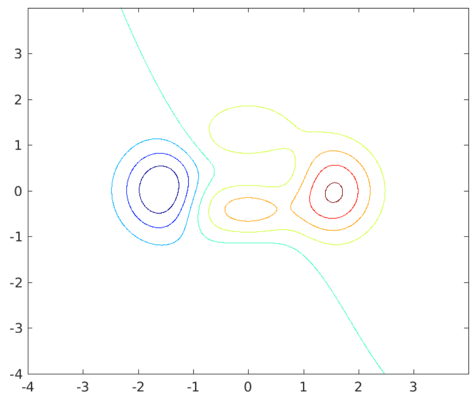
Resources

- `http://infinity77.net/global_optimization/test_functions_nd_P.html`

A.3.36 Peaks function



(a) Plot



(b) Contours

Figure A.33: Peaks function

$$f_{36}(\mathbf{x}) = g_1(\mathbf{x}) - g_2(\mathbf{x}) - g_3(\mathbf{x}) \tag{A.36}$$

where:

$$g_1(\mathbf{x}) = 3\left(1 - x_1\right)^2 e^{\left[-x_1^2 - (x_2+1)^2\right]}$$

$$g_2(\mathbf{x}) = 10\left(\frac{x_1}{5} - x_1^3 - x_2^5\right) e^{\left(-x_1^2 - x_2^2\right)}$$

$$g_3(\mathbf{x}) = \frac{1}{3} e^{\left[-\left(x_1+1\right)^2 - x_2^2\right]}$$

Bounds

$$-4 \le x_i \le 4$$

Optima

$$f_{36,min}(\mathbf{x}^*) = -6.551133332622496$$

$$\mathbf{x}^* \approx (0.228279999979237, -1.625531071954464)$$

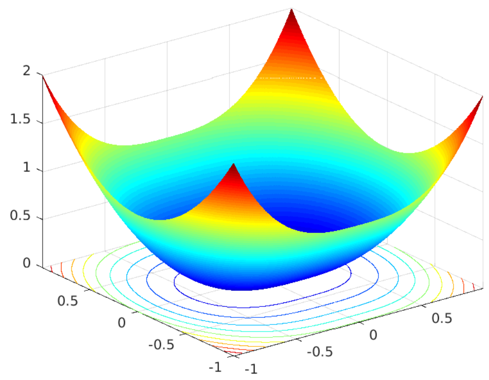
Properties

- 2-dimensional
- Non-separable
- Multimodal

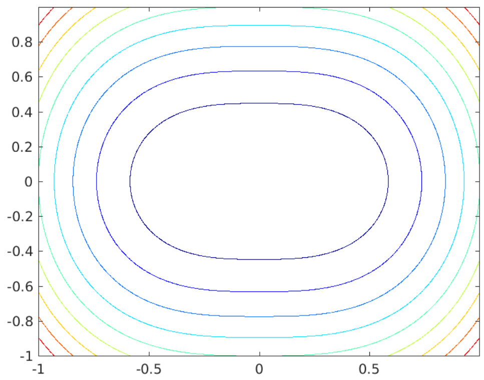
Resources

- `http://al-roomi.org/benchmarks/unconstrained/2-dimensions/63-peaks-function`

A.3.37 Powell sum



(a) Plot



(b) Contours

Figure A.34: Powell sum function

$$f_{37}(\mathbf{x}) = \sum_{i=1}^{n} |x_i|^{i+1} \tag{A.37}$$

Bounds

$$-1 \leq x_i \leq 1$$

Optima

$$f_{37,min}(\mathbf{x}^*) = 0$$
$$x_i^* = 0$$

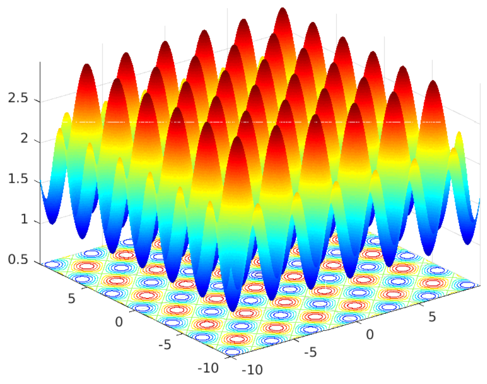
Properties

- n-dimensional
- Separable
- Unimodal

Resources

- `http://benchmarkfcns.xyz/benchmarkfcns/powellsumfcn.html`

A.3.38 Price's Function No.02


(a) Plot

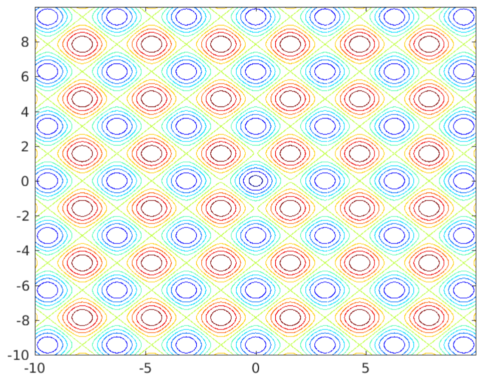

(b) Contours

Figure A.35: Price's Function No.02

$$f_{38}(\mathbf{x}) = 1 + \sin^2(x_1) + \sin^2(x_2) - 0.1e^{\left(-x_1^2-x_2^2\right)} \tag{A.38}$$

Bounds

$$-10 \leq x_i \leq 10$$

Optima

$$f_{38,min}(\mathbf{x}^*) = 0.9$$
$$x_i^* = 0$$

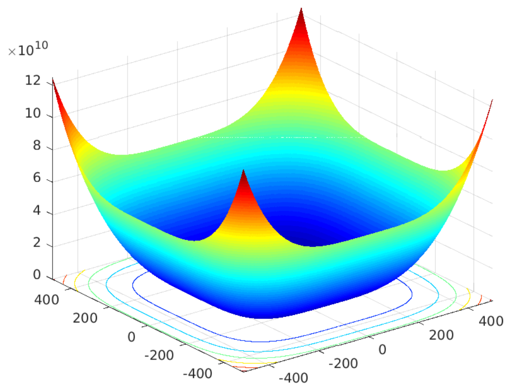
Properties

- 2-dimensional
- Non-Separable
- Multimodal

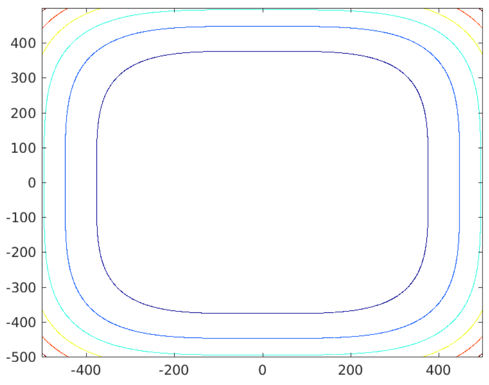
Resources

- `http://al-roomi.org/benchmarks/unconstrained/2-dimensions/158-price-s-function-no-`

A.3.39 Qing's Function



(a) Plot



(b) Contours

Figure A.36: Qing function

$$f_{39}(\mathbf{x}) = \sum_{i=1}^{n} \left(x_i^2 - i\right)^2 \tag{A.39}$$

Bounds

$$-500 \leq x_i \leq 500$$

Optima

$$f_{39,min}(\mathbf{x}^*) = 0$$
$$x_i^* = \pm\sqrt{i}$$

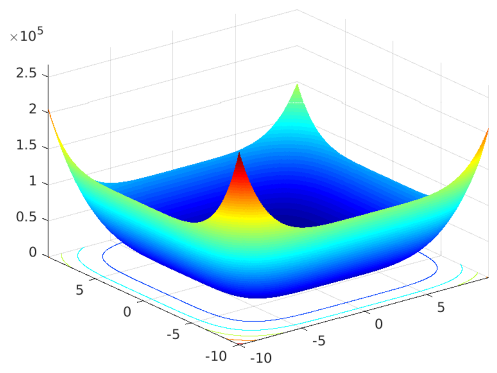
Properties

- n-dimensional
- Separable
- Multimodal

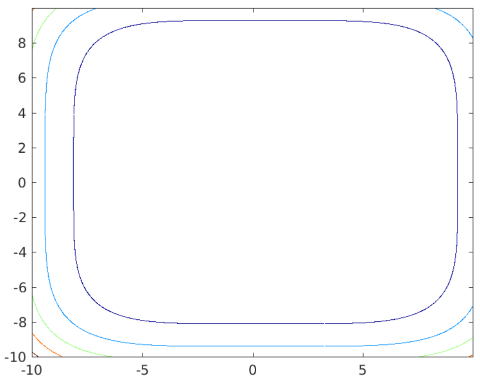
Resources

- `http://al-roomi.org/benchmarks/unconstrained/n-dimensions/185-qing-s-function`

A.3.40 Quintic Function



(a) Plot



(b) Contours

Figure A.37: Quintic Function

$$f_{40}(\mathbf{x}) = \sum_{i=1}^{n} \left| x_i^5 - 3x_i^4 + 4x_i^3 + 2x_i^2 - 10x_i - 4 \right| \tag{A.40}$$

Bounds

$$-10 \leq x_i \leq 10$$

Optima

$$f_{40,min}(X^*) = 0$$
$$\mathbf{x}^* = (-1, 2)$$

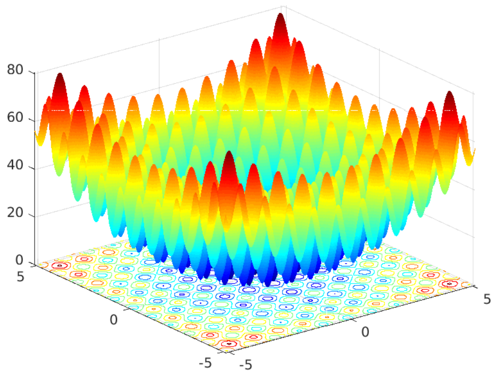
Properties

- n-dimensional
- Separable
- Multimodal

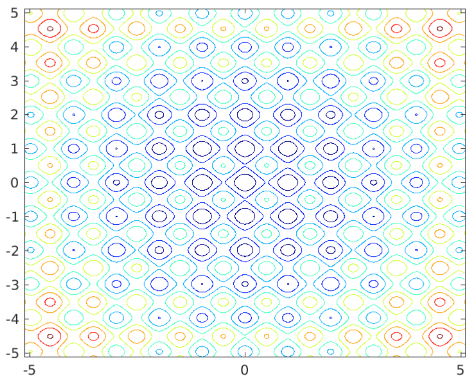
Resources

- `http://al-roomi.org/benchmarks/unconstrained/n-dimensions/240-quintic-function`

A.3.41 Rastrigin function



(a) Plot



(b) Contours

Figure A.38: Rastrigin function

$$f_{41}(\mathbf{x}) = 10n + \sum_{i=1}^{n}(x_i^2 - 10cos(2\pi x_i)) \tag{A.41}$$

Bounds

$$-5.12 \leq x_i \leq 5.12$$

Optima

$$f_{41,min}(\mathbf{x}^*) = 0$$
$$x_i^* = 0$$

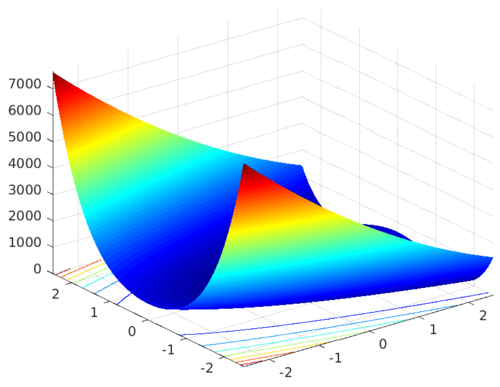
Properties

- n-dimensional
- Non-separable
- Multimodal

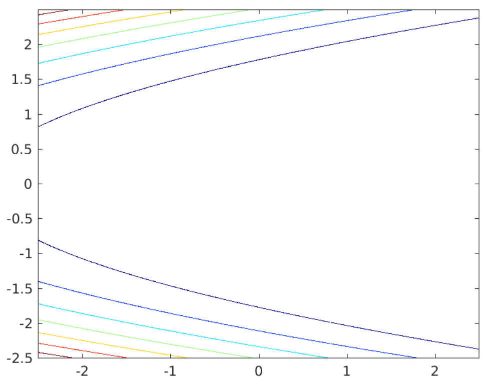
Resources

- `https://en.wikipedia.org/wiki/Rastrigin_function`
- `https://www.sfu.ca/~ssurjano/rastr.html`

A.3.42 Rosenbrocks Function



(a) Plot



(b) Contours

Figure A.39: Rosenbrocks Function

$$f_{42}(\mathbf{x}) = \sum_{i=1}^{n-1}(100(x_i^2 - x_{i+1})^2 + (1 - x_i)^2) \tag{A.42}$$

Bounds

$$-30 \leq x_i \leq 30$$

Optima

$$f_{42,min}(\mathbf{x}^*) = 0$$
$$x_i^* = 1$$

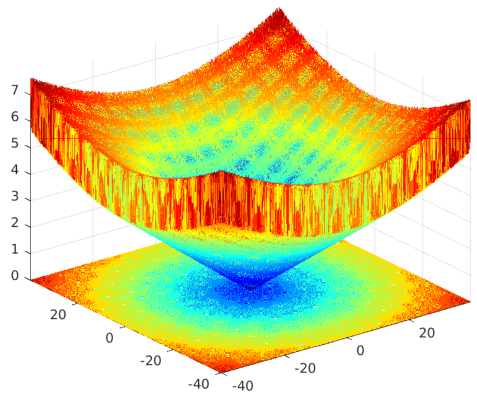
Properties

- n-dimensional
- Non-Separable
- Unimodal

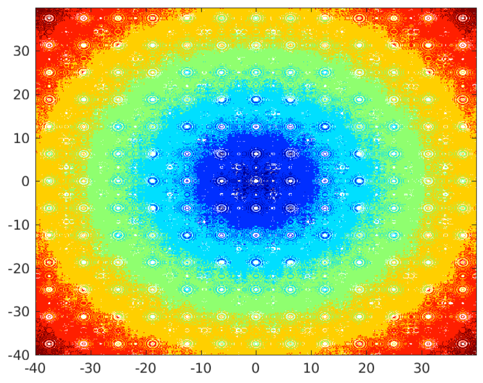
Resources

- `https://en.wikipedia.org/wiki/Rosenbrock_function`
- `https://www.sfu.ca/~ssurjano/rosen.html`

A.3.43 Salomon's Function



(a) Plot



(b) Contours

Figure A.40: Salomon's Function

$$f_{43}(\mathbf{x}) = 1 - \cos\left(2\pi \left\|x\right\|\right) + 0.1 \left\|x\right\| \tag{A.43}$$

with

$$\|x\| = \sqrt{\sum_{i=1}^{n} x_i^2}$$

Bounds

$$-100 \leq x_i \leq 100$$

Optima

$$f_{43,min}(\mathbf{x}^*) = 0$$
$$x_i^* = 0$$

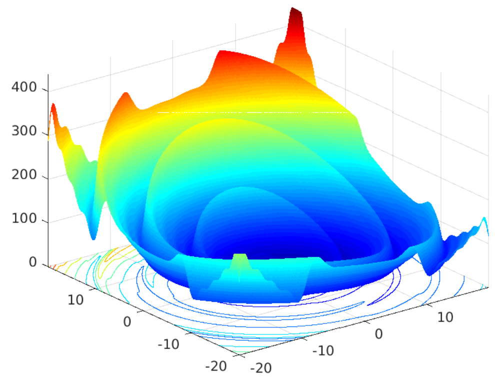
Properties

- n-dimensional
- Non-Separable
- Multimodal

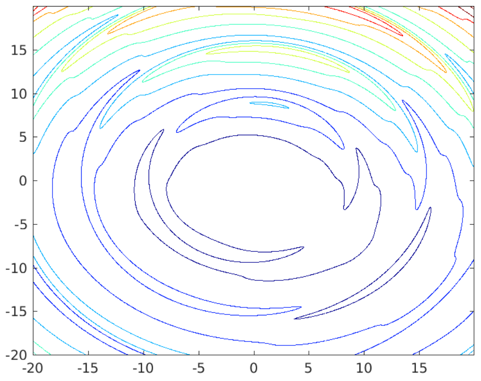
Resources

- `http://al-roomi.org/benchmarks/unconstrained/n-dimensions/184-salomon-s-function`

A.3.44 Sawtoothxy Function



(a) Plot



(b) Contours

Figure A.41: Egg Crate Function

$$f_{44}(\mathbf{x}) = g(r) \cdot h(t) \tag{A.44}$$

$$g(r) = \left[\sin(r) - \frac{\sin(2r)}{2} + \frac{\sin(3r)}{3} - \frac{\sin(4r)}{4} + 4\right]\left(\frac{r^2}{r+1}\right)$$

$$h(t) = \frac{1}{2}\cos\left(2t - \frac{1}{2}\right) + \cos(t) + 2$$

$$r = \sqrt{x_1^2 + x_2^2}$$

$$t = \operatorname{atan2}(x_2, x_1)$$

Bounds

$$-20 \leq x_i \leq 20$$

Optima

$$f_{44,min}(\mathbf{x}^*) = 0$$

$$x_i^* = 0$$

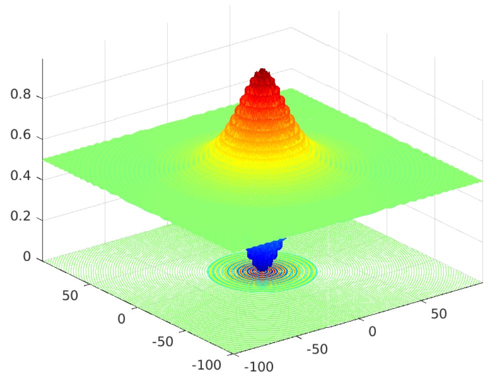
Properties

- 2-dimensional
- Non-Separable
- Multimodal

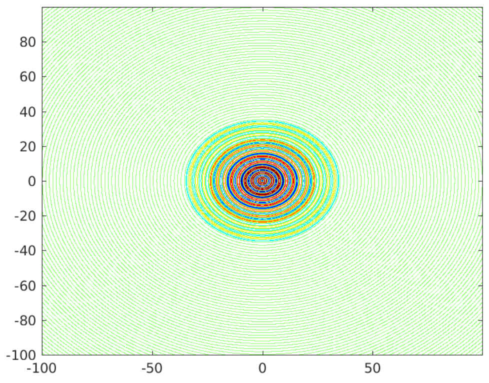
Resources

- `http://al-roomi.org/benchmarks/unconstrained/2-dimensions/66-sawtoothxy-function`

A.3.45 Schaffer's F6



(a) Plot



(b) Contours

Figure A.42: Schaffer's F6

$$f_{45}(\mathbf{x}) = 0.5 + \frac{sin^2(\sqrt{x_1^2 + x_2^2}) - 0.5}{[1 + 0.001 \cdot (x_1^2 + x_2^2)]^2} \tag{A.45}$$

Bounds

$$-100 \leq x_i \leq 100$$

Optima

$$f_{45,min}(\mathbf{x}^*) = 0$$
$$\mathbf{x}^* = (0, 0)$$

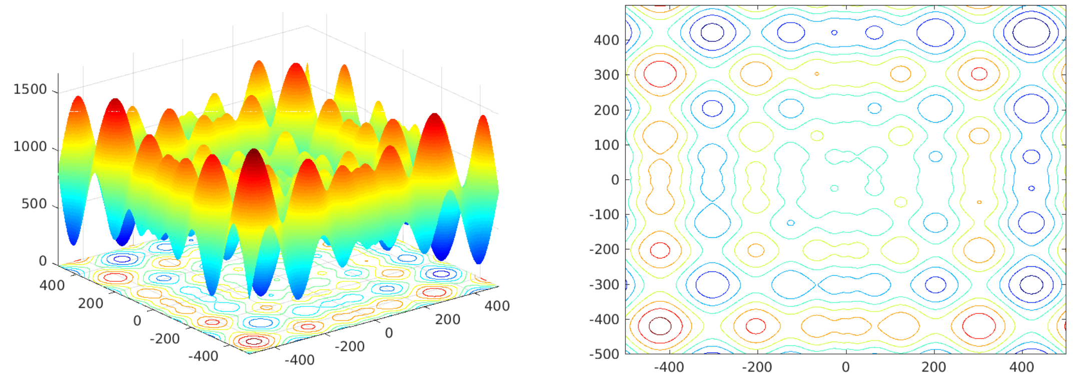
Properties

- 2-dimensional
- Non-Seperable
- Multimodal

Resources

- `http://www.cs.unm.edu/~neal.holts/dga/benchmarkFunction/schafferf6.html`

A.3.46 Schwefel



(a) Plot (b) Contours

Figure A.43: Schwefel function

$$f_{46}(\mathbf{x}) = \sum_{i=1}^{n}(-x_i sin(\sqrt{|x_i|})) + \alpha \cdot n \tag{A.46}$$

$$\alpha = 418.982887$$

Bounds

$$-512 \leq x_i \leq 512$$

Optima

$$f_{46,min}(\mathbf{x}^*) = 0$$

$$x_i^* = 420.968746$$

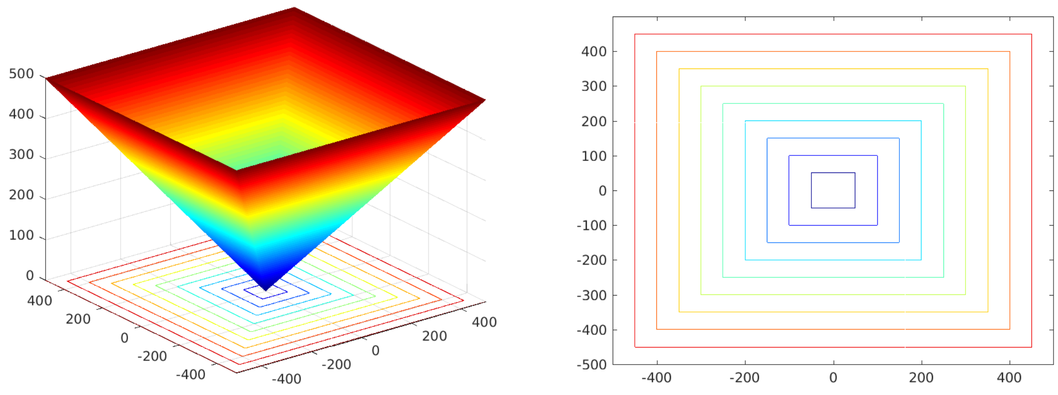
Properties

- n-dimensional
- Separable
- Multimodal

Resources

- `https://www.sfu.ca/~ssurjano/schwef.html`

A.3.47 Schwefel F2.21



(a) Plot (b) Contours

Figure A.44: Schwefel F2.21 Function

$$f_{47}(\mathbf{x}) = \max_i\{|x_i|, 1 \leq i \leq n\} \tag{A.47}$$

Bounds

$$-100 \leq x_i \leq 100$$

Optima

$$f_{47min}(\mathbf{x}^*) = 0$$
$$x_i^* = 0$$

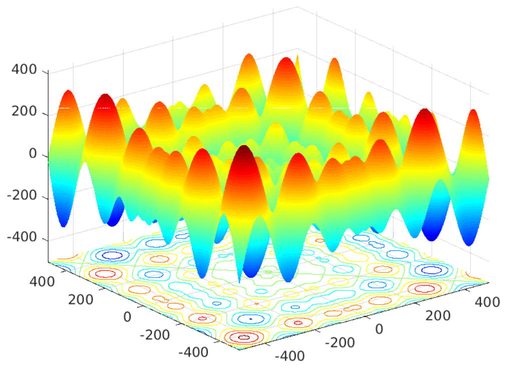
Properties

- n-dimensional
- Separable
- Unimodal

Resources

- `http://al-roomi.org/benchmarks/unconstrained/n-dimensions/189-schwefel-s-function-no-2`

A.3.48 Schwefel F2.26


(a) Plot

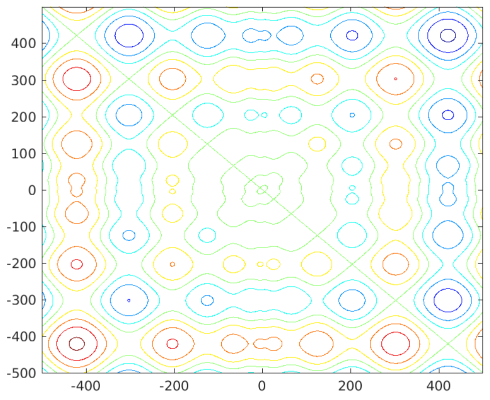

(b) Contours

Figure A.45: Schwefel F2.26 Function

$$f_{48}(\mathbf{x}) = -\frac{1}{n}\sum_{i=1}^{n} x_i \sin\sqrt{|x_i|} \tag{A.48}$$

Bounds

$$-500 \leq x_i \leq 500$$

Optima

$$f_{48,min}(\mathbf{x}^*) = -418.983$$
$$x_i^* = \pm[\pi(0.6+k)]^2$$

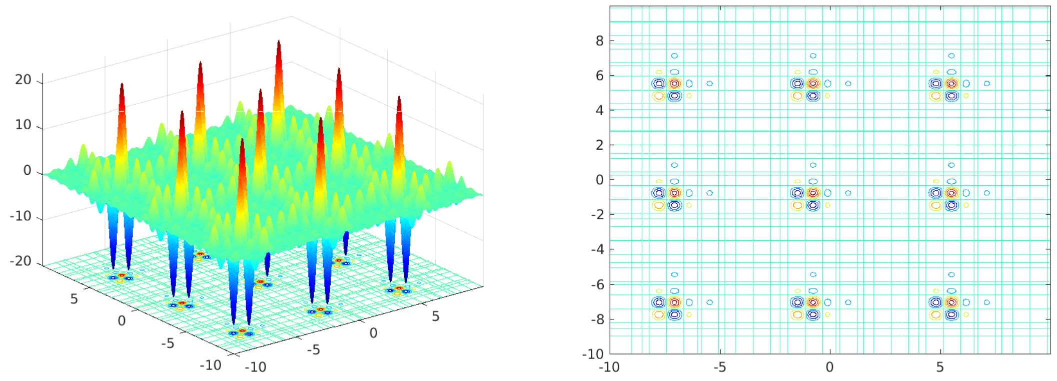
Properties

- n-dimensional
- Separable
- Multimodal

Resources

- `http://al-roomi.org/benchmarks/unconstrained/n-dimensions/176-generalized-schwefel`

A.3.49 Shubert Function



(a) Plot (b) Contours

Figure A.46: Shubert Function

$$f_{49}(\mathbf{x}) = \prod_{i=1}^{n} \left(\sum_{j=1}^{5} cos((j+1)x_i + j) \right) \tag{A.49}$$

Bounds

$$-10 \le x_i \le 10$$

Optima

$$f_{min}(\mathbf{x}^*) = -186.7309$$

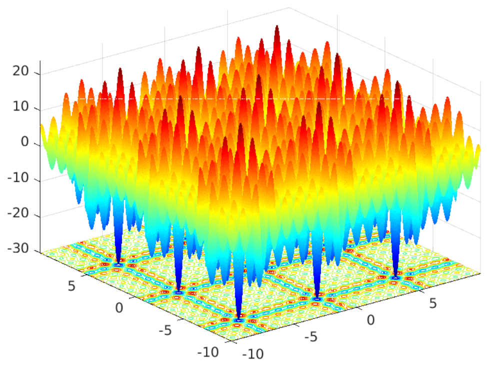
Properties

- n-dimensional
- Separable
- Multimodal

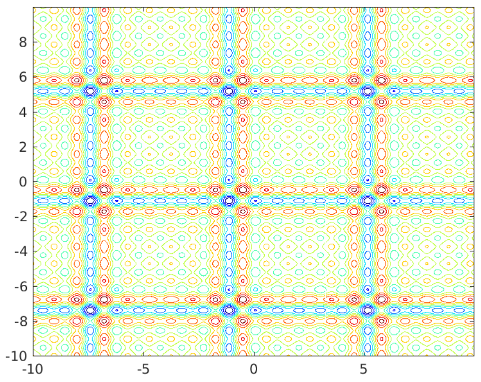
Resources

- `https://www.sfu.ca/~ssurjano/shubert.html`
- `http://infinity77.net/global_optimization/test_functions_nd_S.html`

A.3.50 Shubert 3 Function



(a) Plot



(b) Contours

Figure A.47: Shubert 3 Function

$$f_{50}(\mathbf{x}) = \sum_{i=1}^{n} \sum_{j=1}^{5} j \sin((j+1)x_i + j) \tag{A.50}$$

Bounds

$$-10 \leq x_i \leq 10$$

Optima

$$f_{50,min}(\mathbf{x}^*) = -29.6733337$$

Multiple solutions.

Properties

- n-dimensional
- Separable
- Multimodal

Resources

- `http://infinity77.net/global_optimization/test_functions_nd_S.html`

A.3.51 Shubert 4 Function

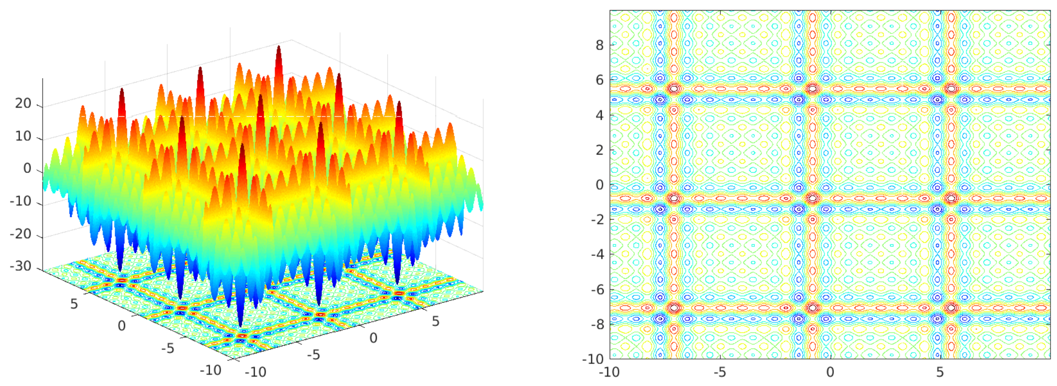

(a) Plot (b) Contours

Figure A.48: Shubert 4 Function

$$f_{51}(\mathbf{x}) = \sum_{i=1}^{n}\sum_{j=1}^{5} j\cos((j+1)x_i + j) \tag{A.51}$$

Bounds

$$-10 \leq x_i \leq 10$$

Optima

$$f_{min}(\mathbf{x}^*) = -25.740858$$

Multiple solutions.

Properties

- n-dimensional
- Separable
- Multimodal

Resources

- `http://infinity77.net/global_optimization/test_functions_nd_S.html`

A.3.52 Six-Hump Camel-Back Function

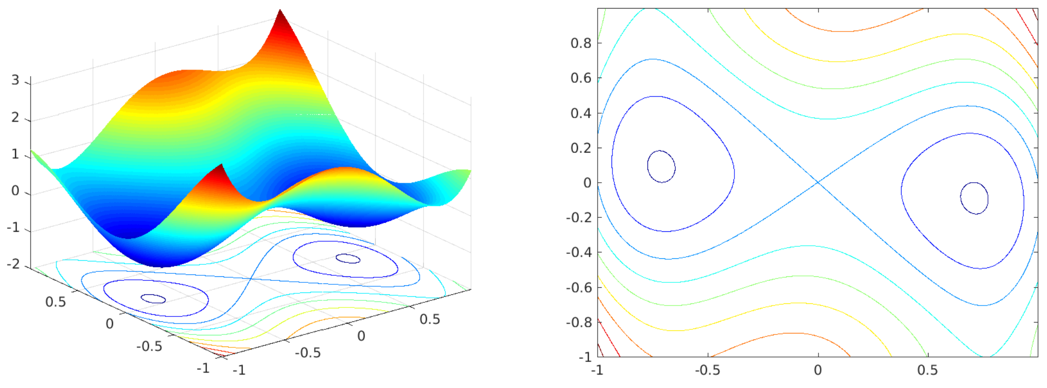

(a) Plot (b) Contours

Figure A.49: Six-Hump Camel-Back Function

$$f_{52}(\mathbf{x}) = 4x_1^2 - 2.1x_1^4 + \frac{1}{3}x_1^6 + x_1x_2 - 4x_2^2 + 4x_2^4 \tag{A.52}$$

Bounds

$$-5 \leq x_i \leq 5$$

Optima

$$f_{52,min}(\mathbf{x}^*) = -1.031628453489877$$

$$\mathbf{x}^* \approx (\pm 0.08984201368301331, \pm 0.7126564032704135)$$

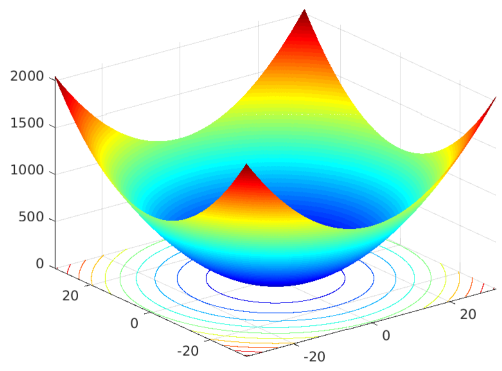
Properties

- 2-dimensional
- Multimodal

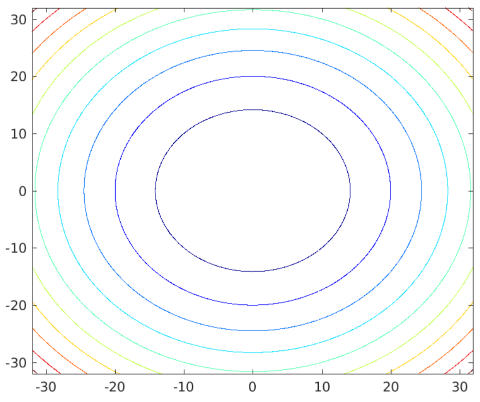
Resources

- `http://infinity77.net/global_optimization/test_functions_nd_S.html`
- `https://www.sfu.ca/~ssurjano/camel6.html`

A.3.53 Sphere function



(a) Plot



(b) Contours

Figure A.50: Sphere function

$$f_{53}(\mathbf{x}) = \sum_{i=1}^{n} x_i^2 \tag{A.53}$$

Bounds

$$-100 \leq x_i \leq 100$$

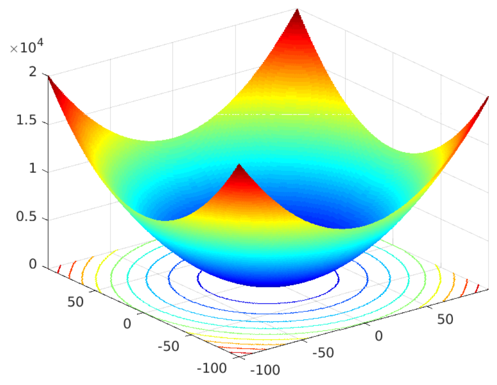
Optima

$$f_{53,min}(\mathbf{x}^*) = 0$$
$$x_i^* = 0$$

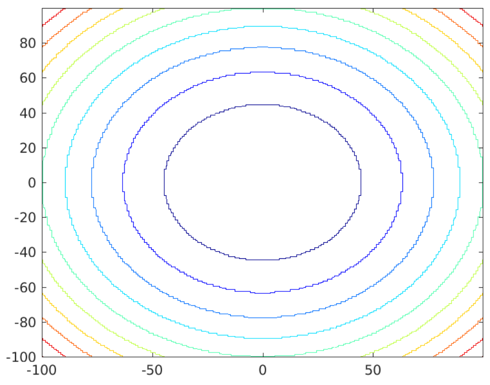
Properties

- n-dimensional
- Separable
- Unimodal

A.3.54 Step Function No.02



(a) Plot



(b) Contours

Figure A.51: Step Function No.02 function

$$f(\mathbf{x}) = \sum_{i=1}^{n} (\lfloor |x_i + 0.5| \rfloor)^2 \tag{A.54}$$

Bounds

$$-100 \leq x_i \leq 100$$

Optima

$$f_{54,min}(\mathbf{x}^*) = 0$$

$$-0.5 \leq x_i^* < 0.5$$

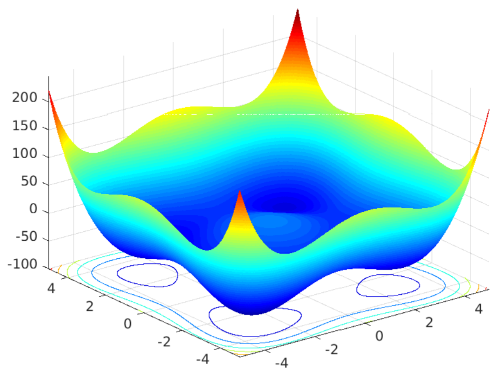
Properties

- n-dimensional
- Separable
- Unimodal

Resources

- `http://al-roomi.org/benchmarks/unconstrained/n-dimensions/193-step-function-no-2`

A.3.55 Styblinski-Tang


(a) Plot

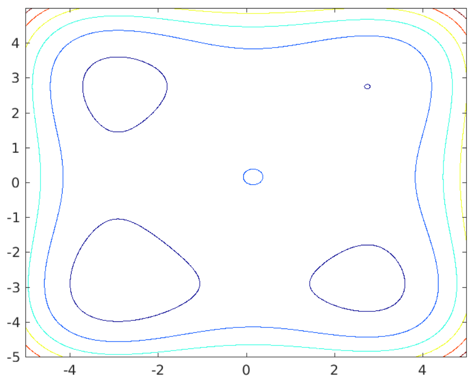

(b) Contours

Figure A.52: Styblinski-Tang

$$f_{55}(\mathbf{x}) = \sum_{i=1}^{n} \left(x_i^4 - 16x_i^2 + 5x_i \right) \tag{A.55}$$

Bounds

$$-5 \leq x_i \leq 5$$

Optima

$$f_{55,min}(\mathbf{x}^*) = -39.16616570377142n$$
$$x_i^* = -2.903534018185960$$

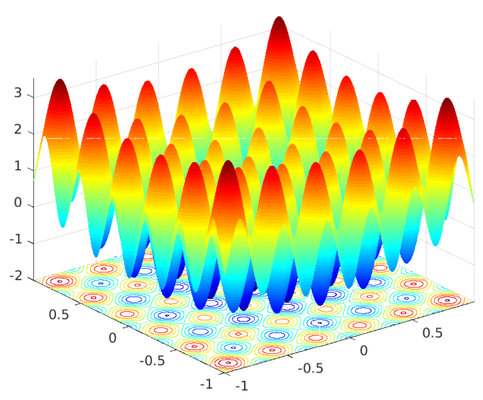
Properties

- n-dimensional
- Non-Separable
- Multimodal

Resources

- `https://www.sfu.ca/~ssurjano/stybtang.html`

A.3.56 Tsoulos' Function


(a) Plot

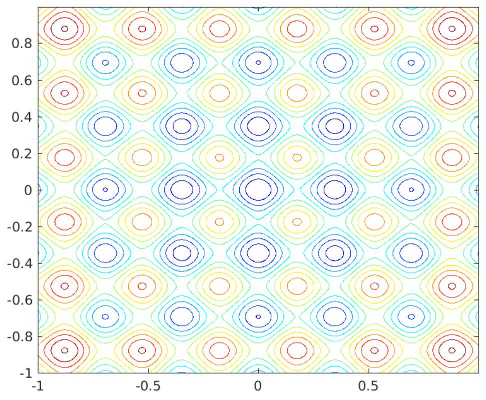

(b) Contours

Figure A.53: Tsoulos' Function

$$f_{56}(\mathbf{x}) = x_1^2 + x_2^2 - \cos(18x_1) - \cos(18x_2) \tag{A.56}$$

Bounds

$$-1 \leq x_i \leq 1$$

Optima

$$f_{56,min}(\mathbf{x}^*) = -2$$
$$\mathbf{x}^* = (0, 0)$$

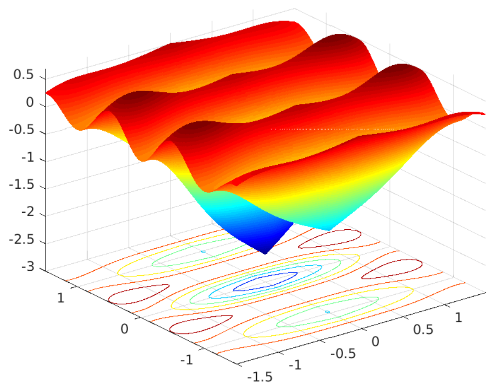
Properties

- 2-dimensional
- Multimodal

Resources

- `http://al-roomi.org/benchmarks/unconstrained/2-dimensions/106-tsoulos-function`

A.3.57 Ursem Function No.03


(a) Plot

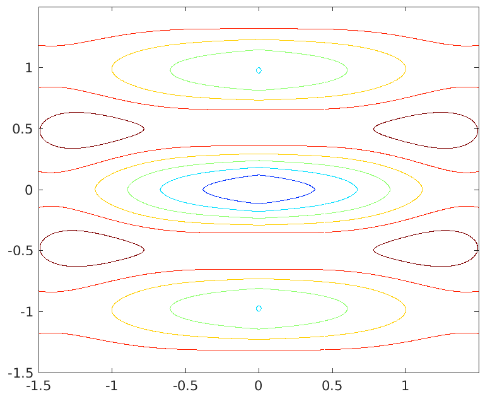

(b) Contours

Figure A.54: Ursem Function No.03

$$f_{57}(\mathbf{x}) = -\frac{3-|x_1|}{2}.\frac{2-|x_2|}{2}.\sin\left(2.2\pi x_1 + 0.5\pi\right) - \frac{2-|x_1|}{2}.\frac{2-|x_2|}{2}.\sin\left(0.5\pi x_2^2 + 0.5\pi\right) \tag{A.57}$$

Bounds

$$-2 \le x_1 \le 2$$
$$-1.5 \le x_2 \le 1.5$$

Optima

$$f_{57,min}(\mathbf{x}^*) = -2.5$$
$$x_i^* = 0$$

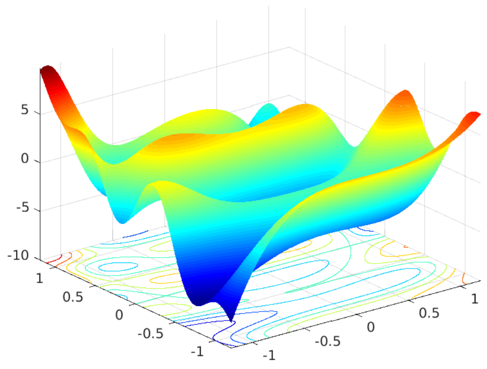
Properties

- 2-dimensional
- Non-separable

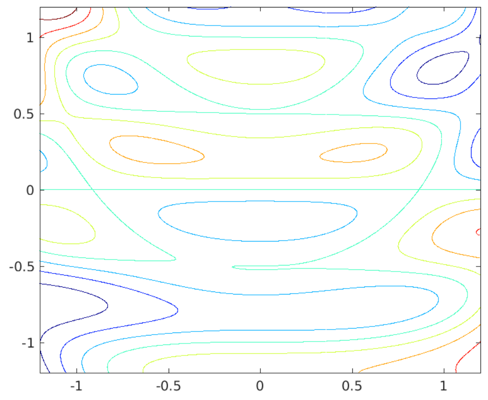
Resources

- `http://al-roomi.org/benchmarks/unconstrained/2-dimensions/130-ursem-function-no-3`

A.3.58 Ursem-Waves Function



(a) Plot



(b) Contours

Figure A.55: Ursem-Waves Function

$$f_{58}(\mathbf{x}) = -\left(0.3x_1\right)^3 + \left(x_2^2 - 4.5x_2^2\right)x_1x_2 + 4.7\cos\left[3x_1 - x_2^2\left(2 + x_1\right)\right]\sin\left(2.5\pi x_1\right) \quad \text{(A.58)}$$

Bounds

$$-0.9 \leq x_1 \leq 1.2, -1.2 \leq x_2 \leq 1.2$$

Optima

$$f_{58,min}(\mathbf{x}^*) = -7.306998731324462$$

$$\mathbf{x}^* = (-0.605689494589848, -1.177561933039789)$$

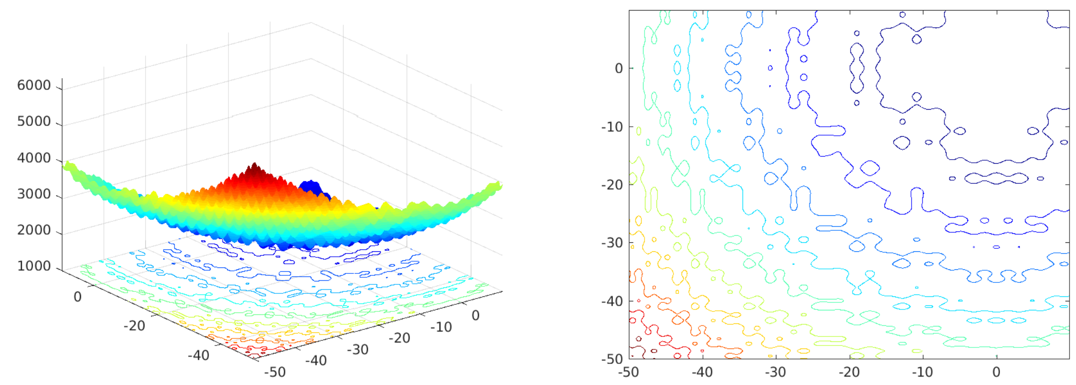
Properties

- 2-dimensional
- Non-separable

Resources

- `http://infinity77.net/global_optimization/test_functions_nd_U.html`
- `http://al-roomi.org/benchmarks/unconstrained/2-dimensions/132-ursem-wave-function`

A.3.59 Venter and Sobiezcczanski-Sobieski's Function



(a) Plot (b) Contours

Figure A.56: Venter Sobiezcczanski-Sobieski function

$$f_{59}(\mathbf{x}) = \sum_{i=1}^{n} \left[x_i^2 - 100\cos^2(x_i) - 100\cos\left(\frac{x_i^2}{30}\right) \right] + 1400 \qquad \text{(A.59)}$$

Bounds

$$-50 \leq x_i \leq 10$$

Optima

$$f_{59,min}(\mathbf{x}^*) = 1000$$
$$x_i^* = 0$$

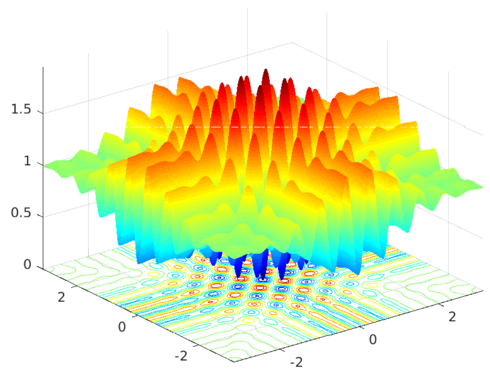
Properties

- 2-dimensional
- Separable

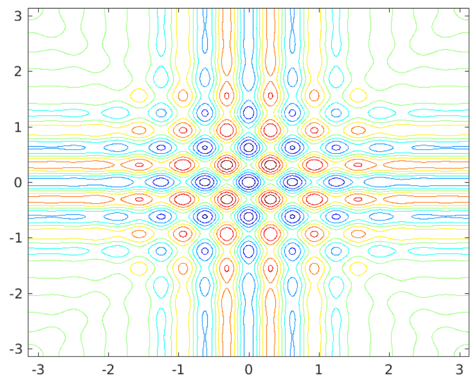
Resources

- `http://al-roomi.org/benchmarks/unconstrained/2-dimensions/108-venter-and-sobiezcczanski`
- `http://infinity77.net/global_optimization/test_functions_nd_V.html`

A.3.60 W / Wavy Function



(a) Plot



(b) Contours

Figure A.57: W / Wavy Function

$$f_{60}(\mathbf{x}) = \frac{1}{n}\sum_{i=1}^{n} 1 - \cos\left(kx_i\right) e^{-\frac{1}{2}x_i^2} \tag{A.60}$$

$$k = 10$$

Bounds

$$-\pi \le x_i \le \pi$$

Optima

$$f_{60,min}(\mathbf{x}^*) = 0$$

$$x_i^* = 0$$

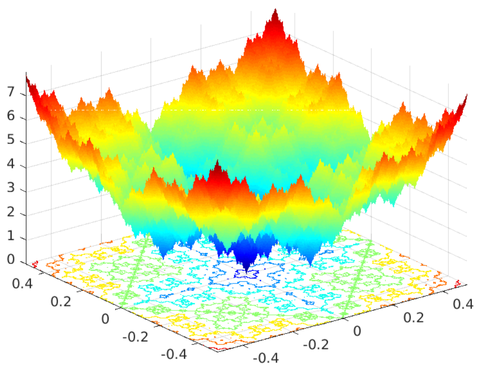
Properties

- n-dimensional
- Separable
- Multimodal

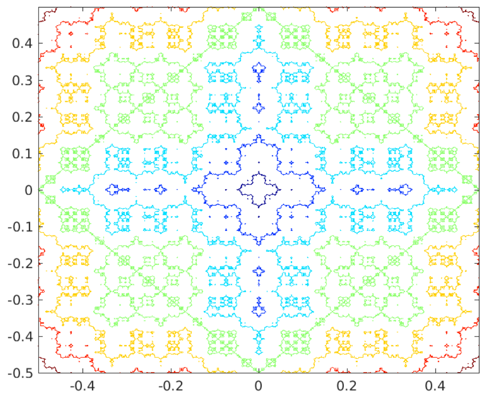
Resources

- `http://al-roomi.org/benchmarks/unconstrained/n-dimensions/280-w-wavy-function`
- `http://infinity77.net/global_optimization/test_functions_nd_W.html`

A.3.61 Weierstrass



(a) Plot



(b) Contours

Figure A.58: Weierstrass function

$$f_{61}(\mathbf{x}) = \sum_{i=1}^{n} \left(\sum_{k=0}^{k_{max}} \left[a^k \cos(2\pi b^k (x_i + 0.5)) \right] - n \sum_{k=0}^{k_{max}} \left[a^k \cos(2\pi b^k \cdot 0.5) \right] \right) \tag{A.61}$$

Bounds

$$-0.5 \leq x_i \leq 0.5$$

Optima

$$f_{61,min}(\mathbf{x}^*) = 0$$
$$x_i^* = 0$$

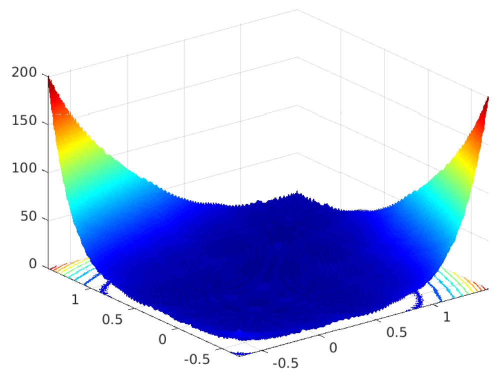
Properties

- n-dimensional
- Separable
- Multimodal

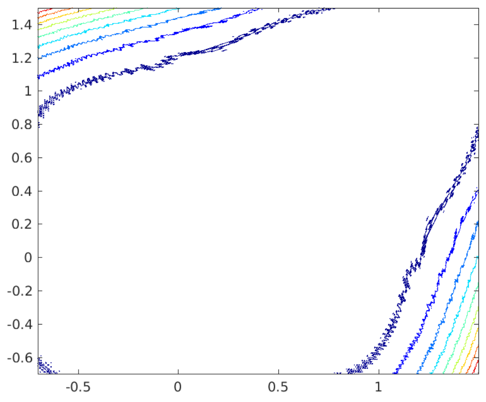
Resources

- `http://infinity77.net/global_optimization/test_functions_nd_W.html`

A.3.62 Whitley



(a) Plot



(b) Contours

Figure A.59: Whitley Function

$$f_{62}(\mathbf{x}) = \sum_{i=1}^{n}\sum_{j=1}^{n}\left[\frac{\left(100\left(x_i^2 - x_j\right)^2 + (1 - x_j)^2\right)^2}{4000} - \cos\left(100\left(x_i^2 - x_j\right)^2 + (1 - x_j)^2 + 1\right)\right] \tag{A.62}$$

Bounds

$$-10.24 \leq x_i \leq 10.24$$

Optima

$$f_{62,min}(\mathbf{x}^*) = 0$$
$$x_i^* = 1$$

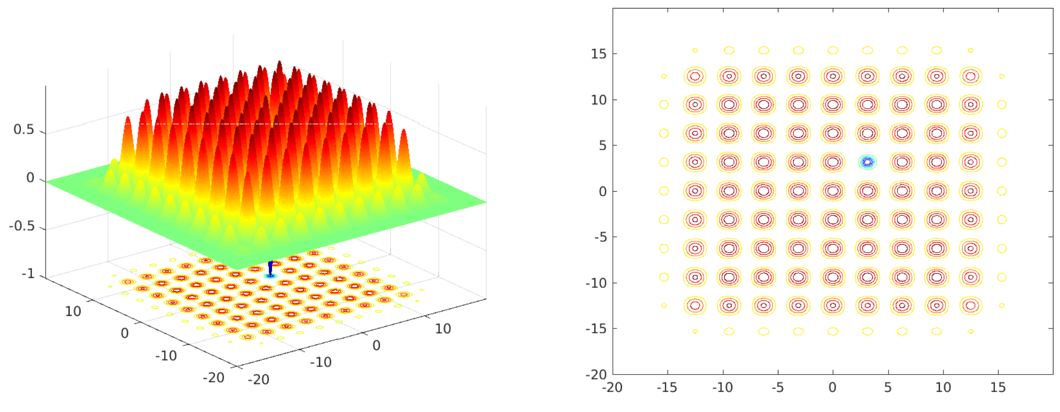
Properties

- n-dimensional
- Non-Separable
- Multimodal

Resources

- `http://infinity77.net/global_optimization/test_functions_nd_W.html`
- `http://www.cs.unm.edu/~neal.holts/dga/benchmarkFunction/whitley.html`

A.3.63 Xin-She Yang's Function No.01



(a) Plot

(b) Contours

Figure A.60: Xin-She Yang's Function No.01

$$f_{63}(\mathbf{x}) = \left[e^{-\sum_{i=1}^{n} \left(\frac{x_i}{\beta}\right)^{2m}} - 2e^{-\sum_{i=1}^{n}(x_i - c)^2} \right] \cdot \prod_{i=1}^{n} \cos^2(x_i) \tag{A.63}$$

$$c \text{ can be any value, usually } c = 0 \text{ or } c = \pi$$

$$m = 5$$

$$\beta = 15$$

Bounds

$$-20 \leq x_i \leq 20$$

Optima

$$f_{63,min}(\mathbf{x}^*) = -1$$

$$x_i^* = c$$

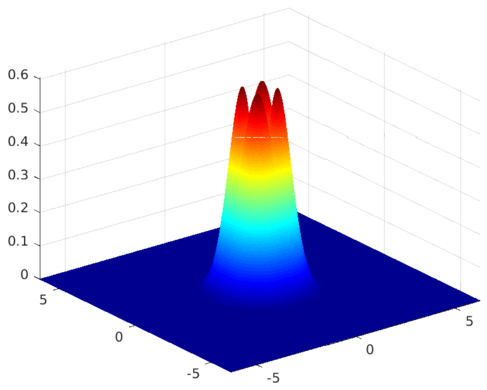
Properties

- n-dimensional
- Non-separable

Resources

- `http://al-roomi.org/benchmarks/unconstrained/n-dimensions/262-xin-she-yang-s-function-`

A.3.64 Xin-She Yang's Function No.02


(a) Plot

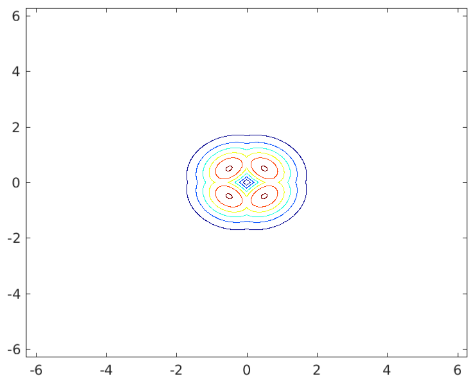

(b) Contours

Figure A.61: Xin-She Yang's Function No.02

$$f_{64}(\mathbf{x}) = \left(\sum_{i=1}^{n} |x_i|\right) \cdot e^{-\sum_{i=1}^{n} x_i^2} \tag{A.64}$$

Bounds

$$-10 \leq x_i \leq 10$$

Optima

For $n = 1$:

$$f_{64,max}(\mathbf{x}^*) = 0.428881942480354$$
$$x^* = \pm 0.707106781903310$$

For $n = 2$:

$$f_{64,max}(\mathbf{x}^*) = \frac{1}{\sqrt{e^1}} = 0.606530659712633$$
$$x_i^* = \pm\frac{1}{2}$$

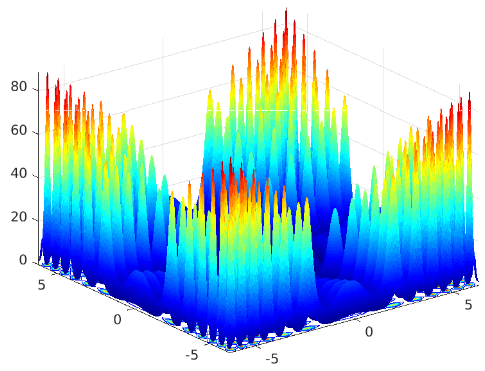
Properties

- n-dimensional

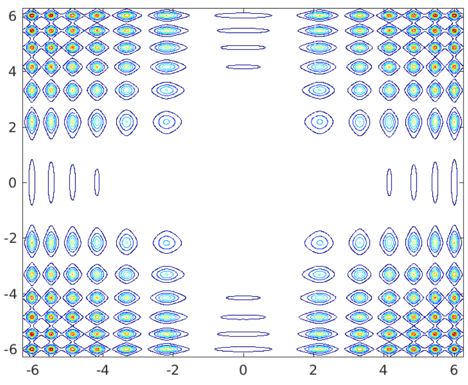
Resources

- `http://al-roomi.org/benchmarks/unconstrained/n-dimensions/264-xin-she-yang-s-func`

A.3.65 Xin-She Yang's Function No.03



(a) Plot



(b) Contours

Figure A.62: Xin-She Yang's Function No.03

$$f_{65}(\mathbf{x}) = \left(\sum_{i=1}^{n} |x_i|\right) \cdot e^{-\sum_{i=1}^{n} \sin\left(x_i^2\right)} \tag{A.65}$$

Bounds

$$-2\pi \leq x_i \leq 2\pi$$

Optima

$$f_{65,min}(\mathbf{x}^*) = 0$$
$$x_i^* = 0$$

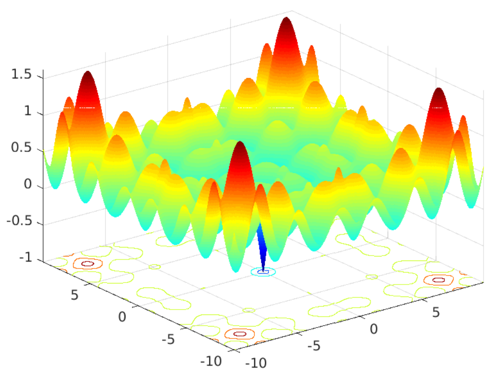
Properties

- n-dimensional

Resources

- `http://al-roomi.org/benchmarks/unconstrained/n-dimensions/263-xin-she-yang-s-function-`

A.3.66 Xin-She Yang's Function No.06


(a) Plot

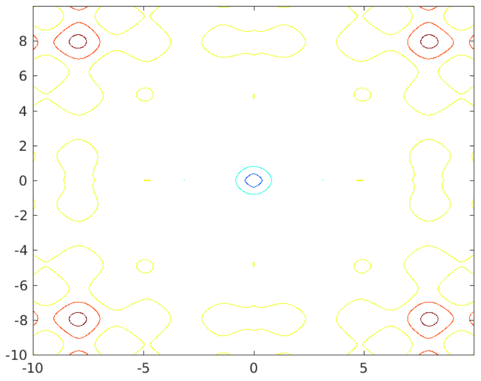

(b) Contours

Figure A.63: Xin-She Yang's Function No.06

$$f_{66}(\mathbf{x}) = \left\{ \left[\sum_{i=1}^{n} \sin^2(x_i) \right] - e^{-\sum_{i=1}^{n} x_i^2} \right\} \cdot e^{-\sum_{i=1}^{n} \sin^2\left(\sqrt{|x_i|}\right)} \tag{A.66}$$

Bounds

$$-10 \le x_i \le 10$$

Optima

$$f_{66,min}(\mathbf{x}^*) = -1$$
$$x_i^* = 0$$

Properties

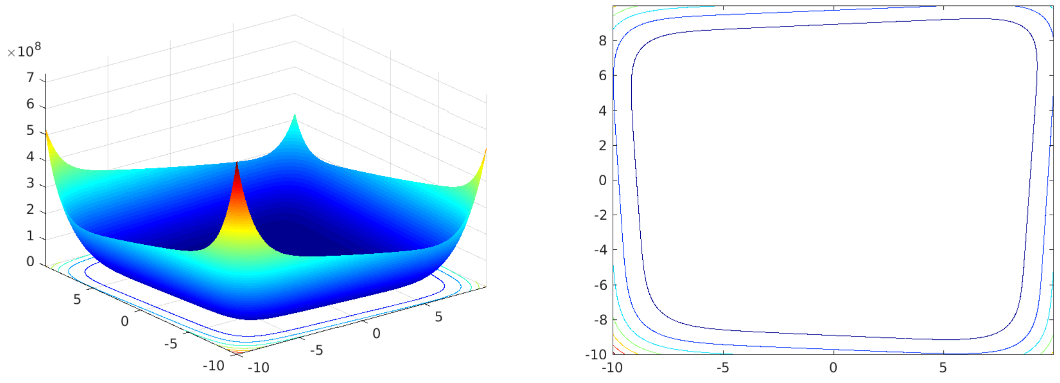
- n-dimensional
- Non-separable

Resources

- `http://al-roomi.org/benchmarks/unconstrained/n-dimensions/268-xin-she-yang-s-func`

A.4 Compound functions

A.4.1 Expanded Griewanks plus Rosenbrocks Function



(a) Plot (b) Contours

Figure A.64: Expanded Griewanks plus Rosenbrocks Function

$$f_{67}(\mathbf{x}) = f_{22}(f_{42}(x_1, x_2)) + f_{22}(f_{42}(x_2, x_3)) + \cdots + f_{22}(f_{42}(x_{n-1}, x_n)) + f_{22}(f_{42}(x_n, x_1)) \tag{A.67}$$

Bounds

$$-100 \leq x_i \leq 100$$

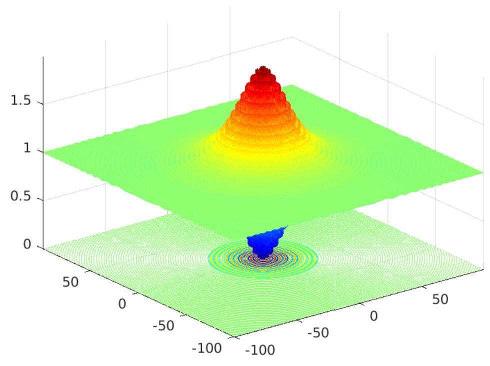
Optima

$$f_{67,min}(\mathbf{x}^*) = 0$$
$$x_i^* = 0$$

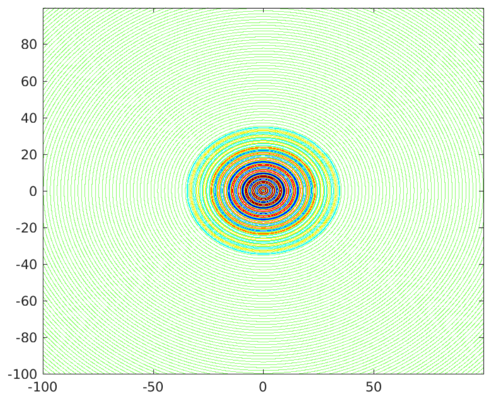
Properties

- n-dimensional
- Non-Separable
- Multimodal

A.4.2 Expanded Scaffers F6 Function



(a) Plot



(b) Contours

Figure A.65: Expanded Scaffers F6 function

$$f_{68}(\mathbf{x}) = f_{45}(x_1, x_2) + f_{45}(x_2, x_3) + \cdots + f_{45}(x_{n-1}, x_n) + f_{45}(x_n, x_1) \tag{A.68}$$

Bounds

$$-100 \leq x_i \leq 100$$

Optima

$$f_{68,min}(\mathbf{x}^*) = 0$$
$$\mathbf{x}^* = (0, 0)$$

Properties

- n-dimensional
- Non-Seperable
- Multimodal

A.4.3 Stacked Bird, Egg-crate, Leon, Sawtoothxy

$$f_{69}(\mathbf{x}) = f_7(x_1, x_2) + f_{19}(x_3, x_4) + f_{28}(x_5, x_6) + f_{44}(x_7, x_8) \tag{A.69}$$

Bounds

$$-2\pi \leq x_1, x_2 \leq 2\pi$$
$$-5 \leq x_3, x_4 \leq 5$$
$$-1.2 \leq x_5, x_6 \leq 1.2$$
$$-20 \leq x_7, x_8 \leq 20$$

Optima

$$f_{69,min}(\mathbf{x}^*) = -106.7645367198034$$
$$\mathbf{x}^* = (4.701055751981055, 3.152946019601391, 0, 0, 1, 1, 0, 0)\,,$$
$$\mathbf{x}^* = (-1.582142172055011, -3.130246799635430, 0, 0, 1, 1, 0, 0)$$

Properties

- n-dimensional
- Non-Seperable
- Multimodal

A.4.4 Stacked Adjiman, Cross in Tray, Crowned cross, Schaffer F6

$$f_{70}(\mathbf{x}) = f_2(x_1, x_2) + f_{12}(x_3, x_4) + f_{13}(x_5, x_6) + f_{45}(x_7, x_8) \tag{A.70}$$

Bounds

$$-5 \leq x_1, x_2 \leq 5$$
$$-15 \leq x_3, x_4 \leq 15$$
$$-10 \leq x_5, x_6 \leq 10$$
$$-100 \leq x_7, x_8 \leq 100$$

Optima

$$f_{70,min}(\mathbf{x}^*) = -x_1^{max} - 2.062611870822739$$
$$\mathbf{x}^* = (x_1^{max}, 0, \pm1.349406608602084, \pm1.349406608602084, 0, 0, 0, 0)$$

Properties

- n-dimensional
- Non-Seperable
- Multimodal

A.4.5 Stacked Davis, Downhill step, Drop-wave, Six-Hump Camel-back

$$f_{71}(\mathbf{x}) = f_{14}(x_1, x_2) + f_{17}(x_3, x_4) + f_{18}(x_5, x_6) + f_{52}(x_7, x_8) \tag{A.71}$$

Bounds

$$-100 \leq x_1, x_2 \leq 100$$
$$-10 \leq x_i \leq 10$$
$$-5.12 \leq x_i \leq 5.12$$
$$-5 \leq x_i \leq 5$$

Optima

$$f_{71,min}(\mathbf{x}^*) = 6.968371547$$
$$\mathbf{x}^* = (0, 0, 0, 0, 0, 0, \pm0.08984201368301331, \pm0.7126564032704135)$$

Properties

- n-dimensional
- Non-Seperable
- Multimodal

A.4.6 Stacked Giunta, Hosaki, Mishra F3, Ursem F3

$$f_{72}(\mathbf{x}) = f_{21}(x_1, x_2) + f_{26}(x_3, x_4) + f_{31}(x_5, x_6) + f_{57}(x_7, x_8) \tag{A.72}$$

Bounds

$$-1 \leq x_1, x_2 \leq 1$$
$$0 \leq x_3, x_4 \leq 10$$
$$-10 \leq x_5, x_6 \leq 10$$
$$-2 \leq x_7 \leq 2$$
$$-1.5 \leq x_8 \leq 1.5$$

Optima

$$f_{72,min}(\mathbf{x}^*) = -4.965992489$$
$$\mathbf{x}^* = (0.4673200277395354, 0.4673200169591304, 4, 2,$$
$$-8.466613775046579, -9.998521308999999, 0, 0)$$

Properties

- n-dimensional
- Non-Seperable
- Multimodal