

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



Bachelor's Project

Heterogeneous Parallel Genetic Algorithm

Jan Černý

Supervisor: Ing. Jiří Kubalík, Ph.D.

Study Programme: Softwarové technologie a management, Bakalářský

Field of Study: Inteligentní Systémy

May 27, 2010

Czech Technical University in Prague
Faculty of Electrical Engineering

Department of Cybernetics

BACHELOR PROJECT ASSIGNMENT

Student: Jan Černý
Study programme: Software Engineering and Management
Specialisation: Intelligent Systems
Title of Bachelor Project: Heterogeneous Parallel Genetic Algorithm

Guidelines:

1. Study parallel genetic algorithms and focus on so-called *Island model GA*.
2. Propose and implement a parallel island model GA that runs either a different GA or the same GA with different parameter setting at each island.
3. Carry out experiments with the proposed parallel GA in order to assess its performance.
4. Statistically evaluate achieved results and compare the algorithm with traditional parallel GA.

Bibliography/Sources: Will be provided by the supervisor.

Bachelor Project Supervisor: Ing. Jiří Kubalík, Ph.D.

Valid until: the end of the winter semester of academic year 2010/2011


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




doc. Ing. Boris Šimák, CSc.
Head

Prague, February 5, 2010

České vysoké učení technické v Praze
Fakulta elektrotechnická

Katedra kybernetiky

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Student: Jan Černý
Studijní program: Softwarové technologie a management
Obor: Inteligentní systémy
Název tématu: Heterogenní paralelní genetický algoritmus

Pokyny pro vypracování:

1. Nastudujte problematiku paralelních genetických algoritmů a zaměřte se na tzv. ostrovní model paralelního genetického algoritmu (*Multiple-deme GA* neboli *Island model GA*).
2. Navrhněte a implementujte paralelní GA s ostrovním modelem, který bude každou populaci vyvíjet pomocí jiného GA nebo pomocí stejného GA ale s jiným nastavením.
3. Experimentálně ověřte funkci tohoto algoritmu na zvolených testovacích úlohách.
4. Dosažené výsledky vyhodnoťte a porovnejte s výsledky klasického paralelního GA, který vyvíjí všechny populace stejným GA.


Seznam odborné literatury: Dodá vedoucí práce.

Vedoucí bakalářské práce: Ing. Jiří Kubalík, Ph.D.

Platnost zadání: do konce zimního semestru 2010/2011


prof. Ing. Vladimír Mařík, DrSc.
vedoucí katedry




doc. Ing. Boris Šimák, CSc.
děkan

V Praze dne 5. 2. 2010

Acknowledgements

I would like to thank my supervisor Ing. Jiří Kubalík, Ph.D. whose guidance and support allowed me to overcome all difficulties and finish this work.

Declaration

I hereby declare that I have completed this thesis independently and that I have used only the sources (literature, software, etc.) listed in the enclosed bibliography.

In Prague on May 27, 2010

.....

Abstract

This thesis compares performance of heterogeneous and homogeneous parallel genetic algorithms for optimization of real functions. For the purpose of this experiment an optimization application capable of both homogeneous and heterogeneous evolution has been created. It has been shown that on several benchmark problems heterogeneous approach give better results while it fails on some others.

Abstrakt

Tato práce srovnává vlastnosti heterogenních a homogenních paralelních genetických algoritmů při optimalizaci reálných funkcí. Pro účely tohoto experimentu byla vytvořena optimalizační aplikace schopná jak homogenní tak i heterogenní evoluce. Bylo ukázáno, že na několika testovacích problémech dosahují heterogenní PGA lepších výsledků zatímco na některých jiných selhávají.

Contents

1	Introduction	1
2	Conventional Evolution Algorithms	3
2.1	Canonical Genetic Algorithm	3
2.2	Chromosome Representation	3
2.2.1	Direct Binary Representation	3
2.2.2	Gray Coded Binary Representation	4
2.2.3	Real Representation	4
2.2.4	IEEE 754 format	4
2.3	Binary-coded Crossover Operators	4
2.3.1	Single Point crossover	4
2.3.2	Two Point Crossover	5
2.3.3	Uniform Crossover	5
2.4	Real-coded Crossover Operators	5
2.4.1	Mean Crossover	5
2.4.2	Blend Crossover Operator (BLX- α)	6
2.4.3	Simulated Binary Crossover	6
2.4.4	Parent Centric Crossover	7
2.5	Mutation Operators	7
2.5.1	Point Mutation	8
2.5.2	Mutation of Real Parameters	8
2.6	Evolution Model	8
2.6.1	Generational Evolution	9
2.6.1.1	Elitism	9
2.6.2	Steady-State Evolution	9
2.6.3	Generalized Generation Gap	10
2.7	Parent Selection Strategies	10
2.7.1	Tournament Selection	10
2.7.2	Stochastic Universal Sampling	10
2.7.3	Greedy Over-selection	11
2.7.4	Truncation Selection	11
2.7.5	g3 Selection	11

3	Island Model Parallel EA	13
3.1	Motivation for Island Model	13
3.2	Basic Island Model	13
3.3	Migration Topologies	14
3.4	Migration Schemes	14
4	Heterogeneous EA Concept	15
4.1	Topology and Migration Parameters	15
4.2	Expected Behaviour	16
5	Realization	17
5.1	Language and Libraries Selection	17
5.1.1	PyGene	17
5.1.2	Pyevolve	17
5.2	Package Description	18
5.2.1	Chromosome	18
5.2.2	Organism	18
5.2.3	Population	18
5.2.3.1	Parent Selection and Evolution Model	19
5.2.3.2	Migration	19
5.2.4	World	19
5.2.5	Benchmark Functions	19
5.3	Experiment Configuration	19
6	Experiments and Results	21
6.1	Problem Description	21
6.1.1	Uni-modal Benchmark Problems	21
6.1.1.1	Sphere Function	21
6.1.1.2	Rosenbrock's Function	21
6.1.1.3	Ellipsoidal Function	22
6.1.1.4	Quartic Function	22
6.1.1.5	Schwefel's Double Sum Function	23
6.1.2	Multi-modal Benchmark Problems	24
6.1.2.1	Rastrigin Function	24
6.1.2.2	Schwefel Function	25
6.1.2.3	Masters's Cosine Wave Function	25
6.1.2.4	Pathological Function	25
6.1.2.5	Stretched V Sine Wave Function	26
6.1.2.6	Michalewics Function	27
6.1.3	Binary Only Benchmark Problems	28
6.1.3.1	DF3 Function	28
6.2	Experimental Configuration	28
6.2.1	Experiments with Float Representation	28
6.2.1.1	PGA-1-real	29
6.2.1.2	PGA-2-real	29
6.2.1.3	PGA-3-real	29

6.2.1.4	PGA-4-real	30
6.2.1.5	PGA-5-real	30
6.2.1.6	PGA-mix-real	30
6.2.2	Experiments with Direct Representation	30
6.2.2.1	PGA-1-bin	31
6.2.2.2	PGA-2-bin	31
6.2.2.3	PGA-3-bin	31
6.2.2.4	PGA-4-bin	32
6.2.2.5	PGA-5-bin	32
6.2.2.6	PGA-mix-bin	32
6.2.3	Experiment with DF3 Function	32
6.2.3.1	PGA-1-df3	33
6.2.3.2	PGA-2-df3	33
6.2.3.3	PGA-3-df3	33
6.2.3.4	PGA-4-df3	33
6.2.3.5	PGA-5-df3	33
6.2.3.6	PGA-mix-df3	34
6.3	Experimental Results	34
6.3.1	Experiments with Real Coded Chromosomes	34
6.3.2	Experiments with Direct Coded Chromosomes	34
6.3.3	df3 Experiment	34
7	Conclusion	41
	Bibliography	43
A	List of Abbreviations	45
B	The Wilcoxon Ran-Sum Test	47
B.1	Calculation	47
B.2	P-value	47
B.3	ρ Statistic	48
C	CD Content	49

List of Figures

3.1	Example topology of <i>Island Model</i> PGA	14
4.1	Uni-directional ring topology of Island Model PGA	15
6.1	Sphere Function	22
6.2	Rosenbrock's Function	22
6.3	Ellipsoidal Function	23
6.4	Quartic Function	23
6.5	Schwefel's double sum	24
6.6	Rastrigin Function	24
6.7	Schwefel Function	25
6.8	Masters's cosine wave function	26
6.9	Pathological Function	26
6.10	Stretched-V Sine Wave Function	27
6.11	Michalewics Function	27

List of Tables

6.1	Values for "deceptive function 3"	28
6.2	Median fitness values for experiments with real coded chromosomes	35
6.3	P-values for experiments with real coded chromosomes	36
6.4	ρ -statistic values for experiments with real coded chromosomes	36
6.5	Median fitness values for experiments with direct coded chromosomes	37
6.6	P-values for experiments with direct coded chromosomes	38
6.7	ρ values for experiments with direct coded chromosomes	38
6.8	Results for experiments with df3 fitness function.	39

List of Algorithms

2.1	Mean Crossover	5
2.2	Generational Evolution	9
2.3	Steady State Evolution	9
2.4	Generalized Generation Gap	10
2.5	Tournament Selection	10
2.6	Stochastic Universal Sampling	11
2.7	Greedy over-selection	11

Chapter 1

Introduction

Genetic algorithms are class of optimization methods inspired by Darwinian evolution and natural selection. They are often described as a global search methods that do not use gradient information. Thus, non-differentiable functions as well as functions with multiple local optima represent classes of problems to which genetic algorithms might be applied. Genetic algorithms, as a weak method, are robust but very general [14]. For that reason they are generally a good choice for black-box optimization or when there is no specialized optimization method for a specific problem.

Part of the biological metaphor used to motivate genetic search is that it is inherently parallel. In natural population thousands of individuals execute in parallel [14]. This thesis deals with subset of parallel genetic algorithms called "Island Model". Its main objective is to explore the possibilities and possible benefits offered by employing different genetic algorithms and parameter variation on each population in island model.

To compare performance of heterogeneous PGA with conventional homogeneous PGA some benchmark problems have to be chosen. While genetic algorithms can solve all sorts of optimization problems, function optimization has been chosen for several reasons. The first reason is that benchmark functions are well described in literature, their properties, location and value of the optimal solution are known in advance and can be in certain degree altered as needed. There are also many different benchmark functions that can prove different capabilities of optimization algorithm. Another reason is that it is possible to use different data representations and easily switch between them. With more data representation there is also a greater number of crossover operators which can be employed and that is advantageous for the objective of this work.

In the next chapter, conventional GAs and their operators will described. Description of classical *Island Model* PGA follows in chapter 3. Chapter 4 describes theoretical concepts of proposed heterogeneous IM PGA with implementation described in subsequent chapter. The rest of this work describes performed experiments and their results.

Chapter 2

Conventional Evolution Algorithms

2.1 Canonical Genetic Algorithm

An implementation of genetic algorithm begins with initial population of (typically random) chromosomes [14]. Those chromosomes are then evaluated by fitness function and this evaluation is used to select parent solutions that will be the base for the new generation¹. The chromosome representation (see section 2.2) and fitness function are usually the only problem dependent parts of genetic algorithms [14]. There are many different ways how to select parents for recombination (see section 2.7), how to create offspring (see section 2.3) and how to perform mutation on them (see section 2.5), some of those that were implemented for the purpose of this work are described in this chapter.

2.2 Chromosome Representation

In every genetic algorithm, one of the basic properties is how the data are represented in chromosomes. The chosen data representation has large impact on the choice of crossover and mutation operators. For the purpose of this experiment, four possible representations of real numbers were implemented.

2.2.1 Direct Binary Representation

With this representation, each number is converted to a binary string and a chromosome is created by connecting all those strings together. For this representation, it's necessary to set the upper and lower boundary and precision (number of bits to represent each number). If x_{up} is the upper boundary, x_{down} is the lower boundary and x_b is the number of bits used to represent each gene then any single part of chromosome can be computed with equation (2.1).

$$x_2 = bin \left((x_{10} - x_{down}) \cdot \frac{2^{x_b} - 1}{x_{down} - x_{up}} \right) \quad (2.1)$$

¹This is not true for some evolution models like *Minimal Generation Gap Model* which select parents randomly but for traditional GA models this principle is quite important

This representation allows binary crossover operators and binary mutation but it has several disadvantages, especially that its not possible to use it to represent a number from outside the pre-set boundaries.

2.2.2 Gray Coded Binary Representation

This representation is very similar to the Direct binary representation, it also represents chromosome as binary string and equation (2.1) is also used to compute chromosome parts, but each part is then transformed to the Gray code before the final chromosome is constructed [15].

2.2.3 Real Representation

With this representation, chromosomes have the same length as the dimension of the problem and each gene contains one real number. This is probably the most natural representation for selected optimization problem but it requires special crossover operators.

2.2.4 IEEE 754 format

IEEE-754 is the standard for floating point arithmetic and it also specifies how floating point numbers are represented in computer memory. IEEE-754 representation creates chromosomes in the form of binary strings identical to those in computer memory. This has the advantage that no precision is lost and it is not necessary to set any boundaries. On the contrary, there is a disadvantage that this format is not optimal for genetic algorithms since the number is represented as mantissa and exponent and it has a fixed binary length of 64 bits [5].

2.3 Binary-coded Crossover Operators

The following three crossover operators can be used on any of the three binary representation.

2.3.1 Single Point crossover

Single point binary crossover is the most basic crossover operation for genetic algorithms. It takes two parent solutions and creates two offspring. The algorithm used to create offspring can be described as follows:

1. select crossover point in parent chromosome
2. split both parent chromosomes into two parts at this crossover point
3. create child by combining first part from the first parent and second part from the second parent
4. create second child by combining first part from the second parent and second part from the first parent

2.3.2 Two Point Crossover

Two point binary crossover is quite similar to single point binary crossover, the difference is that it uses two crossover points. The algorithm can be described as:

1. select two crossover points
2. split both parent chromosomes into three parts at those two crossover points
3. create the first child from the first and the third section of the first parent and the middle section of the second parent
4. create the second child from the first and the third section of the second parent and the middle section of the first parent

2.3.3 Uniform Crossover

The uniform crossover differs from the previous two crossover operators in the fact that there is no given number of crossover points, but each bit is treated separately. This operator produces two children from two parents. The process of creating children chromosomes is rather simple: two parent chromosomes are copied and each corresponding bit is swapped between them with probability of 0.5.

2.4 Real-coded Crossover Operators

The following four crossover operators are applied to real values of chromosomes.

2.4.1 Mean Crossover

This is one of the simplest crossover operators for real coded chromosomes. It creates one child from two parents and this child is located between them. The algorithm is quite simple [2.1](#).

Algorithm 2.1 Mean Crossover

```

1:  $A \leftarrow \text{parent1 chromosome}$ 
2:  $B \leftarrow \text{parent2 chromosome}$ 
3: for  $i = 0$  to  $\text{chromosome length}$  do
4:    $\text{child}[i] = (A[i] + B[i])/2$ 
5: end for

```

It can be easily seen that each new generation created by this crossover operator occupies smaller area in search space and if the optimal solution is outside this area it is impossible for this operator to find it. On the other hand this crossover operator can be quite effective on simpler problems if bounds of optimal solution are known and are within space covered by initial population.

2.4.2 Blend Crossover Operator (BLX- α)

This crossover operator creates a single child solution ($x^{(1,t+1)}$) from two parent solutions ($x^{(1,t)}, x^{(2,t)}$). When u_i is random number from 0 to 1, child solution is computed using equation (2.2) [3].

$$x_i^{(1,t+1)} = (1 - \gamma_i)x_i^{(1,t)} + \gamma_i x_i^{(2,t)} \quad (2.2)$$

$$\gamma_i = (1 + 2\alpha)u_i + \alpha \quad (2.3)$$

If α is 0 this generates a random solution in range ($x^{(1,t)}, x^{(2,t)}$).

Equation 2.2 can be written as:

$$\left(x_i^{(1,t+1)} - x_i^{(1,t)}\right) = \gamma_i \left(x_i^{(2,t)} - x_i^{(1,t)}\right) \quad (2.4)$$

From this form it is easy to see that the location of child solution depends on the distance of parent solutions. If the distance between parent solutions is small then the distance between the first parent and child will also be small [3].

2.4.3 Simulated Binary Crossover

This crossover operator was created to simulate the behaviour of the single point binary crossover directly on real parameters in sense that common interval schemata between parents are preserved in children [2].

The procedure of creating children ($x^{(1,t+1)}, x^{(2,t+1)}$) from parent solutions ($x^{(1,t)}, x^{(2,t)}$) can be described as follows [3]: A spread factor is defined as absolute ratio of children difference to that of parents (2.5).

$$\beta_i = \left| \frac{x_i^{(2,t+1)} - x_i^{(1,t+1)}}{x_i^{(2,t)} - x_i^{(1,t)}} \right| \quad (2.5)$$

First, a random number u_i between 0 and 1 is generated. From Specified probability distribution function, the ordinate β_{q_i} is found so that area under the probability curve from 0 to β_{q_i} is equal to u_i . The probability distribution is derived to have similar search power to that of the single point binary crossover and is given as [2]:

$$\Gamma(\beta_i) = \begin{cases} 0.5(\eta + 1)\beta_i^\eta & \text{if } \beta_i \leq 1 \\ 0.5(\eta + 1)\frac{1}{\beta_i^{\eta+2}} & \text{otherwise} \end{cases} \quad (2.6)$$

Using Equation (2.6) we calculate β_{q_i} as:

$$\beta_{q_i} = \begin{cases} (2u_i)^{\frac{1}{\eta+1}} & \text{if } u_i \leq 0.5 \\ \left(\frac{1}{2(1-u_i)}\right)^{\frac{1}{\eta+1}} & \text{otherwise} \end{cases} \quad (2.7)$$

With β_{q_i} the two children solutions are calculated from equations (2.8)(2.9):

$$x^{(1,t+1)} = 0.5[(1 + \beta_{q_i})x^{(1,t)} + (1 - \beta_{q_i})x^{(2,t)}] \quad (2.8)$$

$$x^{(2,t+1)} = 0.5[(1 - \beta_{q_i})x^{(1,t)} + (1 + \beta_{q_i})x^{(2,t)}] \quad (2.9)$$

So the following three steps are required to create children solutions:

1. Generate random number $u_i \in [0, 1]$
2. Calculate β_{q_i} from equation (2.7)
3. Calculate children solutions from equations (2.8)(2.9)

2.4.4 Parent Centric Crossover

Parent centric crossover is a multi-parent crossover operator. Its operation can be described as follows [7, 10]:

1. Select μ parents from population.
2. Compute mean vector \vec{g} from selected parents.
3. Select one random parent $\vec{x}^{(p)}$ as a base for offspring.
4. Compute the direction vector $\vec{d}_p = \vec{x}^{(p)} - \vec{g}$.
5. From the remaining parents compute their perpendicular distances D_i to line $\vec{d}^{(p)}$.
6. \bar{D} is computed as the mean value of all \vec{d}_p .
7. Offspring is calculated from equation (2.10)

$$\vec{y} = \vec{x}^{(p)}\omega_\zeta|\vec{d}^{(p)}| + \sum_{i=1, i \neq p}^{\mu} \omega_\eta \bar{D} \vec{e}^{(i)} \quad (2.10)$$

In equation (2.10) $\vec{e}^{(i)}$ are the $(\mu - 1)$ orthonormal bases that span the subspace perpendicular to $\vec{d}^{(p)}$. The parameters ω_ζ and ω_η are normally distributed variables with variance σ_ζ^2 and σ_η^2 .

The important attribute of parent centric operator is that offspring solutions are centred around parent solutions. Also the distance between parent and offspring depends on the distances between parents.

2.5 Mutation Operators

Mutation is a genetic operator used to enhance genetic diversity and introduce new or previously lost genetic information into the population by changing chromosome of selected individuals.

2.5.1 Point Mutation

Point Mutation operator in GA is based on biological point mutation (or single base substitution). The mutation process is rather simple, with set probability the value of one or more bits is changed to its negation. Therefore this mutation operator works only on chromosomes with bit string representation and can't be used on real represented parameters.

2.5.2 Mutation of Real Parameters

The fact is that when some crossover operators for real parameter chromosomes (for example PCX) are used, mutation is not required since the crossover operator itself adds random variables to the offspring. In spite of that, real parameter mutation is still important for proper function of genetic algorithms. Since the point mutation is not suitable for real parameter mutation, another mutation operator has to be used.

For the purpose of this experiment, the following mutation mechanism has been implemented:

1. When mutation should occur, single variable x is randomly selected from chromosome to be mutated.
2. Several individuals from from population are selected and mean vector \vec{g} is computed from them.
3. Distance σ between mutated individual and mean vector \vec{g} in the direction of variable x is computed.
4. New value is calculated using equation (2.11)

$$y = x + \text{sign}(\vec{g}_x - x)(|\omega| + \sigma) \quad (2.11)$$

The parameter ω from equation (2.11) is a normally distributed variable with variance σ .

This approach has two important positive attributes. The first is that the extent of mutation depends on the actual state of the population (assuming that representative sample has been used to calculate \vec{g}) so there is smaller chance that it would cause negative effects. The Second is that it generates values outside of the space where most crossover operators place offspring, so it can effectively maintain diversity of population.

2.6 Evolution Model

To influence the behaviour of GA in order to adapt it to a specific problem that is to be solved, different evolution models can be employed.

Algorithm 2.2 Generational Evolution

```
1: Create initial population.
2: loop
3:   repeat
4:     select parents
5:     create offspring from selected parents
6:   until new generation is complete
7:   replace parent generation with a new one
8: end loop
```

2.6.1 Generational Evolution

Generational evolution model is one of the most widely used models adopted directly from biology. The algorithm 2.2 is quite straightforward and easy to understand, it also performs quite well on wide variety of problems.

Whit this model, every generation is discreet and offspring does not mix with their parents.

2.6.1.1 Elitism

It is easy to see from algorithm 2.2 that all individuals from parental generation are lost when new generation is finished and there is no guarantee that the best individual from new generation will exceed the old one so it is often desirable to preserve the best genes for the new generation without crossover or mutation. This process is called elitism and is done by simply copying one ore more selected individuals to the new generation before crossovers begin. While this process can save good solutions that would normally be lost it may also disrupt evolution if too many individuals are moved to the new generation.

2.6.2 Steady-State Evolution

The contrary to generational evolution is steady state model described with algorithm 2.3. With steady state there are no discreet generations and children chromosomes are placed between their parents (thus no elitism is needed). It's also possible to tune the behaviour of this model by choosing specific scheme for selection of parent which is removed in step 5 but most widely used and almost always sufficient policy is to replace the worst individual [12].

Algorithm 2.3 Steady State Evolution

```
1: Create initial population.
2: loop
3:   select parents
4:   create offspring from selected parents
5:   replace selected individual with offspring
6: end loop
```

2.6.3 Generalized Generation Gap

Generalized generation gap or g3 model is one of several models designed specifically for function optimization. It differs significantly from conventional models and has no precedence in biology. The algorithm 2.4 is also more complicated but it allows proper function of parent centric crossover (see section 2.4.4) and several others statistic-based operators [7].

Algorithm 2.4 Generalized Generation Gap

```

1: create initial population
2: loop
3:   select best parent and  $\mu - 1$  random parents
4:   create  $\lambda$  offspring from chosen  $\mu$  parents
5:   choose two parents at random from  $\mu$  chosen parents
6:   from combined sub-population of chosen two parents and  $\lambda$  offspring select two best
     solutions to replace selected two parents
7: end loop
  
```

G3 model looks similar to steady state model at first but the most significant differences are in the facts that fitness has minimal role in parent selection and that large number of offspring is created with each crossover operation, depending on used CX operator and optimization problem it might be advisable to set λ to values as high as 200 [7].

2.7 Parent Selection Strategies

To influence the overall behaviour but especially the convergence rate, different parent selection strategies can be used.

2.7.1 Tournament Selection

Tournament selection is quite commonly used selection method 2.5. The idea of this method is to choose randomly some number of individuals and then select the best one of them [4]. This allows for selection pressure to be easily tuned because in large tournaments weaker individuals has much smaller chance of success.

Algorithm 2.5 Tournament Selection

```

1: repeat
2:   select  $k$  (tournament size) individuals at random
3:   select the best individual out of  $k$  individuals
4: until specified number of parents is selected
  
```

2.7.2 Stochastic Universal Sampling

Stochastic Universal Sampling (SUS) is a simple, single phase, $O(N)$ sampling algorithm. It is zero biased, has minimum spread and will achieve all N samples in a single traversal [1].

It can be described as a roulette wheel selection but with equally spaced pointers. It is described in algorithm 2.6.

Algorithm 2.6 Stochastic Universal Sampling

```

1: sort population by fitness
2:  $F \leftarrow$  total fitness of the population
3:  $N \leftarrow$  number of individuals to be selected
4:  $ptr \leftarrow rand(F/N)$ 
5:  $sum \leftarrow 0$ 
6:  $i \leftarrow 0$ 
7: while  $i < N$  do
8:    $sum += population[i].fitness$ 
9:   while  $sum > ptr$  do
10:     $ptr += F/N$ 
11:    select(population[i])
12:     $i += 1$ 
13:   end while
14: end while

```

2.7.3 Greedy Over-selection

Greedy over-selection is a selection mechanism that helps to preserve diversity in population. The process of selecting parents is described by algorithm 2.7.

Algorithm 2.7 Greedy over-selection

```

1: sort population by fitness
2: place best 20% of chromosomes in group H
3: place rest of the chromosomes in group L
4: repeat
5:   80% of the time select individual from group H and 20% from group L
6: until specified number of parents is selected

```

2.7.4 Truncation Selection

Truncation is quite simple artificial selection method in which individuals are sorted from the best to the worst and n best individuals are selected for breeding.

2.7.5 g3 Selection

This selection mechanism selects the best individual and $\mu - 1$ random individuals from population. It is intended for use in g3 evolution model (see section 2.6.3) and there is probably no good reason to use it anywhere else.

Chapter 3

Island Model Parallel EA

3.1 Motivation for Island Model

Part of the biological metaphor used to motivate genetic search is that it is inherently parallel. In natural population thousands of individuals execute in parallel [14]. While there are many ways to exploit this parallelism, in this work the *Island Model* is used.

An *Island Model* genetic algorithm can be described as classical genetic algorithm braked into sub-populations or as multiple genetic algorithms running in parallel and exchanging data. While at first this may look like the main reason for this is to exploit specific hardware configuration by allowing the algorithm to run on multiple processors simultaneously, the fact is that *Island Model* has different advantage.

When there are several isolated genetic algorithms without migration, independent search occurs at each. Each search will be different since the initial population will impose certain sampling bias; also, genetic drift will tend to drive those population in different directions [14]. When migration is introduced, this model is able to exploit differences in all sub-populations and use it as a source of genetic diversity.

3.2 Basic Island Model

The basic design of *Island Model* consists of several sub-populations of the same evolution algorithms with identical parameters. At the beginning the only thing that makes difference between islands is their initial population. Only that and random factor in parent selection and mutation (if there is any) makes it possible for various islands to develop a different scope of genetic material.

The number of populations in *Island Model* PGA is not set nor restricted. In implementations that exploit hardware parallelism, number of islands is commonly set to be the same as number of processors or cluster nodes [14], but in any case the number can vary from two to hundreds of islands [11].

3.3 Migration Topologies

Since number of islands can be greater than number of individuals on them it is sometimes not desirable to exchange individuals between all pairs of islands. For that reason, islands can be aligned into topologies and migration then occurs only between connected islands. One example of such topology is on figure 3.1. This example consists of six islands aligned in circle where every island can directly communicate with its nearest and second nearest neighbours but not with island across from itself.

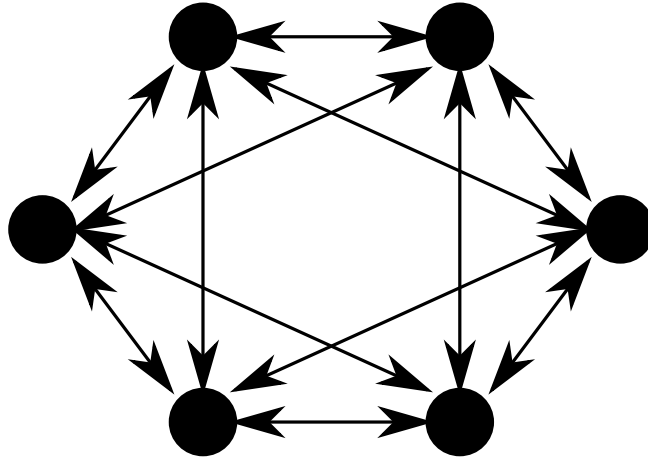


Figure 3.1: Example topology of *Island Model* PGA

3.4 Migration Schemes

While migration is positive for the function of *Island Model* PGA, it can also have negative effects. If the migration is too large then local differences between islands are diminished. When it's too small, it may not be able to prevent local populations from prematurely converging [14].

While the migrations topology can be used to alter the behaviour of *Island Model* PGA it is not sufficient measure. Other parameters that can be used to alter migration are [11]:

Migration Rate determines how many individuals migrate from one population to another.

Migration Frequency determines how often migration occurs.

Migration Policy sets the method used to select individuals for migration.

To reduce the negative effects of migration it is necessary to alter those parameters.

Chapter 4

Heterogeneous EA Concept

In this work a heterogeneous *Island Model* is used. The main difference against the traditional island model is that each island in this model uses different genetic algorithm or at least different settings than other islands.

4.1 Topology and Migration Parameters

While in ordinary island model the migration network topology can be used to alter the speed by which genetic information spreads across all populations, this could have undesired side-effect in heterogeneous model. For example with unidirectional ring topology such as that shown on figure 4.1 genetic information from any population is altered by crossover and mutation operators of for islands before it arrives to island which is one step in clockwise direction. In some cases like when one island is configured to use mean crossover operator (see section 2.4.1) a loss of information will occur which would be avoided with different topology. For that reason a fully connected network topology where each island can communicate with each other will be used.

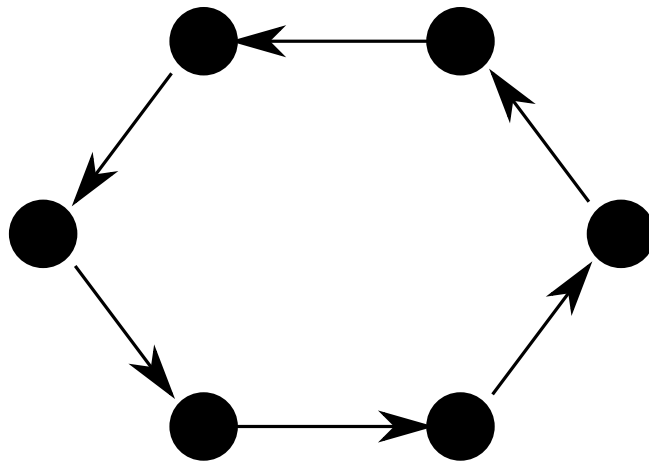


Figure 4.1: Uni-directional ring topology of Island Model PGA

Since complex migration topologies have unwanted side-effects on heterogeneous version of *Island Model* PGA all migration control must be done with migration frequency and migration rate parameters. This alone would not be ideal for large number of islands [11] but it should be sufficient for this work where only small number of islands is used.

4.2 Expected Behaviour

For ordinary single population GAs the choice of genetic operators such as mutation or crossover is determining to their exploration and exploitation powers and predetermines them for certain types of optimization problems while it handicaps them for other problems. Heterogeneous PGA model should exploit different properties of its component population to achieve good results on various optimization problems.

It should also have same capability to preserve genetic variability as homogeneous *Island Model* PGA. In fact islands heterogeneous model should have greater tendency to diverge because of their different genetic operators and thus maintain genetic variability much better than ordinary *Island Model* PGA.

Given all the above heterogeneous PGA should perform better or at least not worse on majority of testing problems.

Chapter 5

Realization

5.1 Language and Libraries Selection

For implementation of this experiment, several programming languages were considered, including C, C++, Go, JAVA and Python. As each of those languages has its advantages and disadvantages, Python was eventually selected for its object oriented properties, reflection, type safety and expressiveness. Python has also full support for unicode strings and automated memory management. The main disadvantage of Python is its speed, from listed languages it is the only one with no compile time optimization¹ and runtime optimization is on lower level than that of JAVA or Go.

There are also several genetic algorithm frameworks and libraries written for python that could have been used in this work. Pyevolve and PyGene frameworks were initially considered as base for this work but in the end none of them were used for various reasons.

5.1.1 PyGene

PyGene is simple and easily understandable framework for genetic algorithms and genetic programming. While it allows for fully customisable recombination, mutation and selection, there is no support for *Island Model* or any other parallel genetic algorithm.

5.1.2 Pyevolve

Pyevolve is more complex, more actively developed and larger framework and it has some very interesting features, for example Interaction module which allows for easy population inspection. In the end Pyevolve has been rejected for the same reason as PyGene, there is no support for any form of parallel evolution yet.

¹While compile time optimization is in development for Python version 3.2, it has not yet been ready at the time of writing of this work

5.2 Package Description

The core of the whole experiment is PyPGA, pure python package which implements all used crossover and mutation operators, selection strategies, evolution models and chromosome representations. The package can be logically divided into four parts:

- Chromosome
- Organism
- Population
- World

There is also a variety of other utilities and benchmark functions but those are not essential parts of PyPGA package.

5.2.1 Chromosome

All chromosome related classes and functions are located in file *genome.py*. This part of PyPGA package handles different chromosome representation described in section 2.2. To provide access to different representations of chromosome there is a variable for each implemented representation. To assure that all those variables always return the same value (in their specific representation) only one value is stored and other variables are set as its transparent encapsulation. This has the advantage that different CX and mutation operators can be used on the same individual without any need to explicitly convert chromosomes every time different representation is needed. On the other hand, this approach brings some performance penalties.

5.2.2 Organism

Each chromosome object belongs to one organism object. Class *Organism* is defined in file *organism.py* and contains methods for all implemented crossover operators described in sections 2.3 and 2.4. The crossover methods require a list of partners for mating, all two parent crossover operators will use the first parent from the list even if the list is longer but multi-parent CX operators generally don't work with just one partner.

Another task that is handled by *Organism* class is fitness evaluation. The reference to fitness function must be supplied as an argument to constructor, when fitness value is needed this function will be called with real-coded chromosome as parameter.

5.2.3 Population

Organisms are grouped in populations. Classes and methods related to population are located in file *population.py* and are responsible for parent selection (see section 2.7), evolution models (see section 2.6) and migration (see section 3.4).

5.2.3.1 Parent Selection and Evolution Model

Both parent selection mechanism and evolution model are set when *Population* object is created according to values supplied to constructor as arguments. It is not possible to use external functions, only methods implemented in class *Population* can be used.

5.2.3.2 Migration

This experiment uses *Island Model* (see section 3.4) parallel genetic algorithm. The migration starts every turn just before parent selection and reproduction, and is implemented so that all migrating individuals are migrated concurrently. In other words, no island can receive individual that originated from there in the same generation and one individual can be migrated to more than one island if both ask for it.

To change the behaviour of migration process several parameters can be set:

Migration Period is used to allow migration to specific island only every n -th turn.

Immigration Count sets number of individuals that will be accepted to the island every time migration happens.

Emigration Probability changes probability that one island will be selected as a source in migration process.

5.2.4 World

All populations in each experiment are grouped into *World* (file *world.py*). The *World* class has no special function other than to keep track of all populations in experiment and assure their synchronization.

5.2.5 Benchmark Functions

The PyPGA is designed so that any function that takes list of numbers as its argument and returns one number can be used as fitness functions. PyPGA package provides several fitness functions (including all functions described in section 6.1) in file *fitness.py*. To use those prepared functions it is sufficient to import *PyPGA.fitness* or single function from this file.

5.3 Experiment Configuration

To run an experiment using PyPGA package it is necessary to write a simple python script that does following:

1. Creates new object of class *World*

2. Adds one or more islands into the newly created world
3. Tells the world to run specified number of generations

The following code sample shows how to create simple *World* with two islands named "a" and "b" and let it evolve for 10 generations. In the end, fitness of the best individuals from both islands is printed to standard output.

```
1 #!/usr/bin/env python3.1
2
3 from PyPGA.world import World
4 from PyPGA.fitness import rosenbrock
5
6 def main():
7     experiment = World(genome_len=5,fitness_function=rosenbrock)
8     experiment.add_island("a", cx_operator="blx_a",
9                           select_function='overselection',
10                          migration_period = 3)
11     experiment.add_island("b", cx_operator="blx_a",
12                              select_function='overselection',
13                              migration_period = 1)
14
15     res = experiment.step(10)
16     print(str(res[0])+"\t"+str(res[1]))
17
18 if __name__ == '__main__': main()
```

Chapter 6

Experiments and Results

6.1 Problem Description

To test the performance of parallel heterogeneous genetic algorithms, several well known and widely used optimization benchmark problems were selected. Those benchmark problems should represent wide variety of properties and several levels of difficulty. For convenience reasons all problems were stated to be minimized.

6.1.1 Uni-modal Benchmark Problems

6.1.1.1 Sphere Function

Sphere function is continuous convex uni-modal function with optimum of zero at the origin. It is defined by equation (6.1). Its two dimensional form is shown on figure 6.1. Because of its simplicity and symmetry, it provides easily analysable first test [6, 8].

$$F1(x) = \sum_{i=1}^D x_i^2 \quad (6.1)$$

6.1.1.2 Rosenbrock's Function

Rosenbrock's function 6.2 is standard test used in optimization literature. It is defined by equation (6.2) and is continuous, non-convex, uni-modal, quadratic function with minimum of zero at (1,1). This function is a difficult optimization problem because of its deep parabolic valley [6, 8].

$$F2(x, y) = 100(x^2 - y)^2 + (x - 1)^2 \quad (6.2)$$

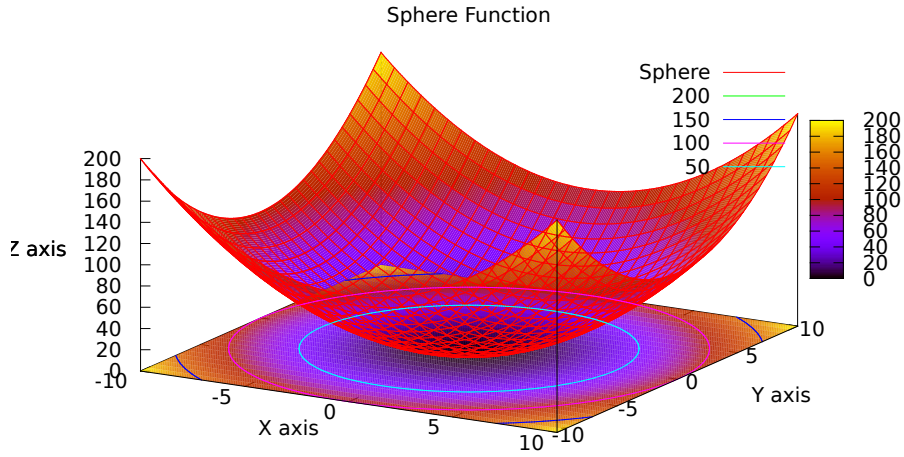


Figure 6.1: Sphere Function

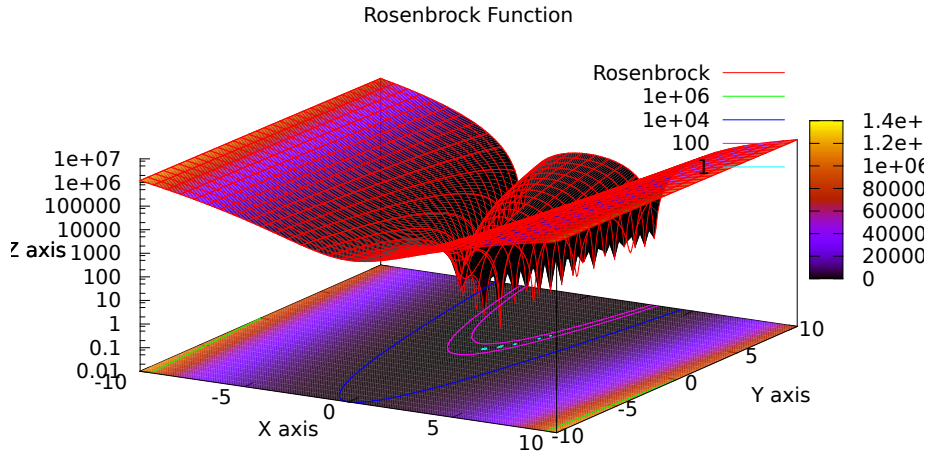


Figure 6.2: Rosenbrock's Function

6.1.1.3 Ellipsoidal Function

Ellipsoidal function 6.3 is a continuous convex uni-modal function defined by (6.3) [8].

$$F3(x) = \sum_{i=1}^D i \cdot x_i \quad (6.3)$$

6.1.1.4 Quartic Function

Quartic function 6.4 is a continuous convex uni-modal function defined by equation (6.4) [6].

$$F9(x, y) = x^4 + 2y^4 \quad (6.4)$$

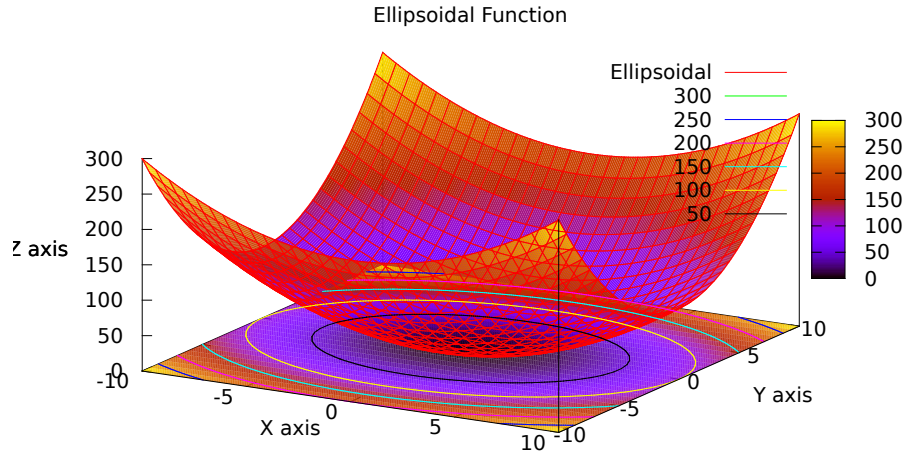


Figure 6.3: Ellipsoidal Function

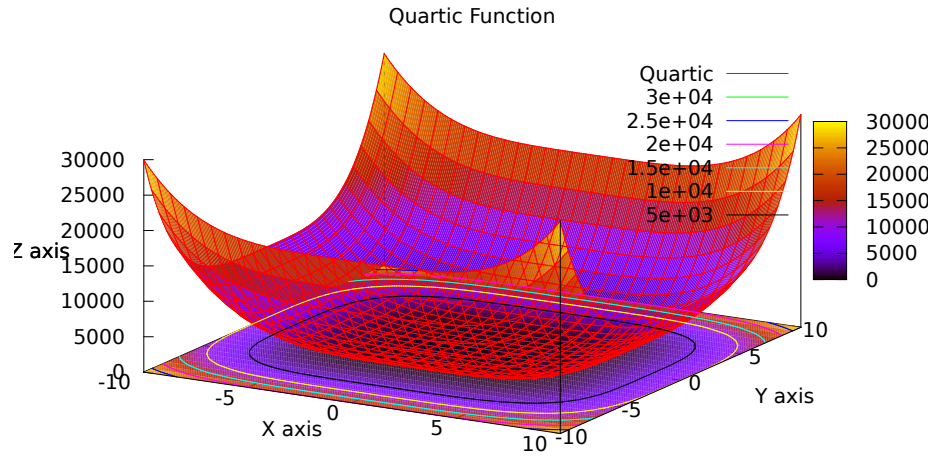


Figure 6.4: Quartic Function

6.1.1.5 Schwefel's Double Sum Function

Schwefel's double sum [6.5](#) functions main difficulty is that its gradient is not oriented along their axis due to the epistasis among their variables; in this way, the algorithms that use the gradient converge very slowly [\[9, 8\]](#). Schwefel's double sum is defined by equation [\(6.5\)](#).

$$F11(x) = \sum_{i=1}^D \left(\sum_{j=1}^i x_j \right) \quad (6.5)$$

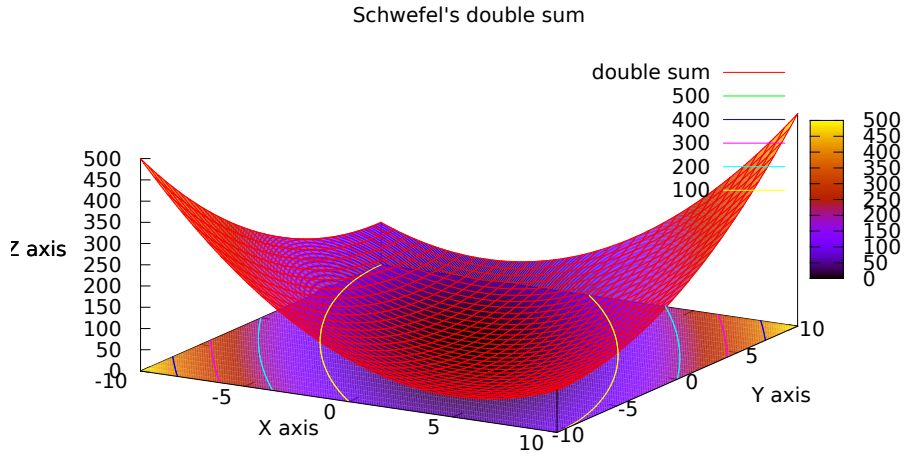


Figure 6.5: Schwefel's double sum

6.1.2 Multi-modal Benchmark Problems

6.1.2.1 Rastrigin Function

Rastrigin function 6.6 was constructed from a sphere adding a modulator term $\alpha \cos(2\pi x_i)$. It has large number of global minima whose value increases with distance from global minimum [9, 8]. The function is defined by equation (6.6).

$$F4(x) = 10D + \sum_{i=1}^D x^2 - 10 \cos(2\pi x) \quad (6.6)$$

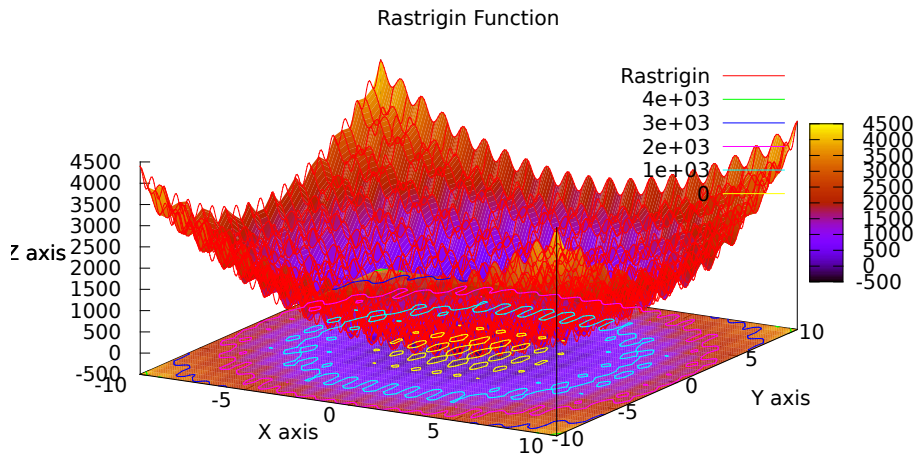


Figure 6.6: Rastrigin Function

6.1.2.2 Schwefel Function

The surface of Schwefel function 6.7 is composed of a great number of peaks and valleys. The function (6.7) has a second best minimum far from the global minimum where many search algorithms are trapped. Moreover, the global minimum is near the bounds of the domain [9, 8].

$$F5(x) = 4.189828872724339 * D + \sum_{i=1}^D x \sin \sqrt{|x|} \quad (6.7)$$

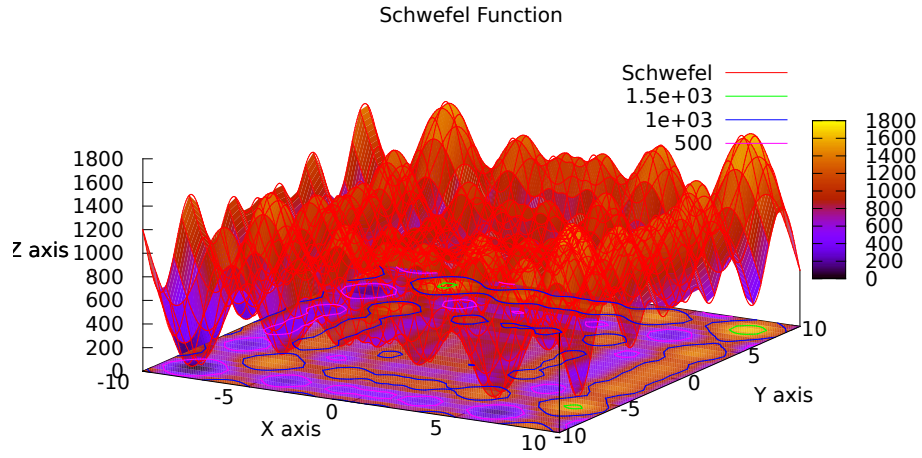


Figure 6.7: Schwefel Function

6.1.2.3 Masters's Cosine Wave Function

Masters's cosine wave function 6.8 is a multi-modal function defined with equation (6.8).

$$F6(x, y) = e^{-\frac{1}{8}(x^2 - 0.5xy + y^2)} \cos(4\sqrt{x^2 - 0.5xy + y^2}) \quad (6.8)$$

6.1.2.4 Pathological Function

Pathological function 6.9 is a multi-modal function defined with equation (6.9).

$$F7(x) = \sum_{i=1}^{D-1} \frac{\sin^2(\sqrt{100x_i^2 + x_{i+1}^2}) - 0.5}{(0.001(x_i^2 - 2x_i x_{i+1} + x_{i+1}^2))^2 + 1} + 0.5 \quad (6.9)$$

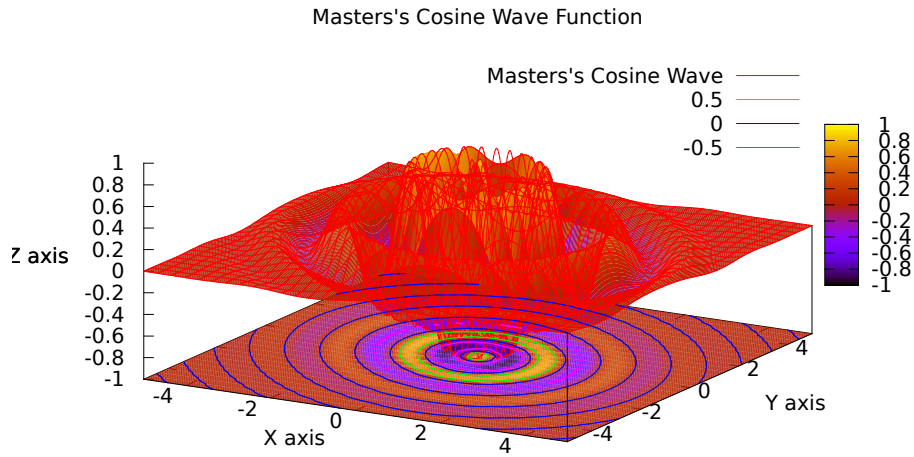


Figure 6.8: Masters's cosine wave function

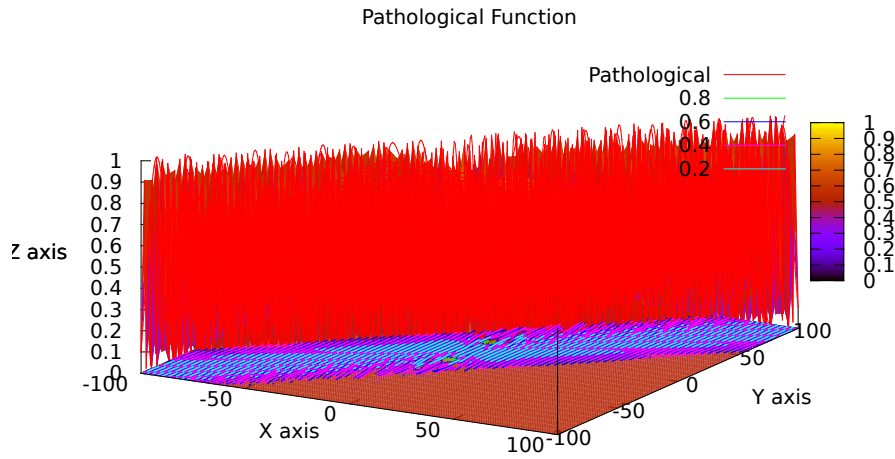


Figure 6.9: Pathological Function

6.1.2.5 Stretched V Sine Wave Function

Stretched V Sine Wave Function 6.10 is defined by equation (6.10).

$$F8(x) = \sum_{i=1}^{D-1} (x_i^2 + x_{i+1}^2)^{0.25} (\sin^2(50(x_i^2 + x_{i+1}^2)^{0.1}) + 1) \quad (6.10)$$

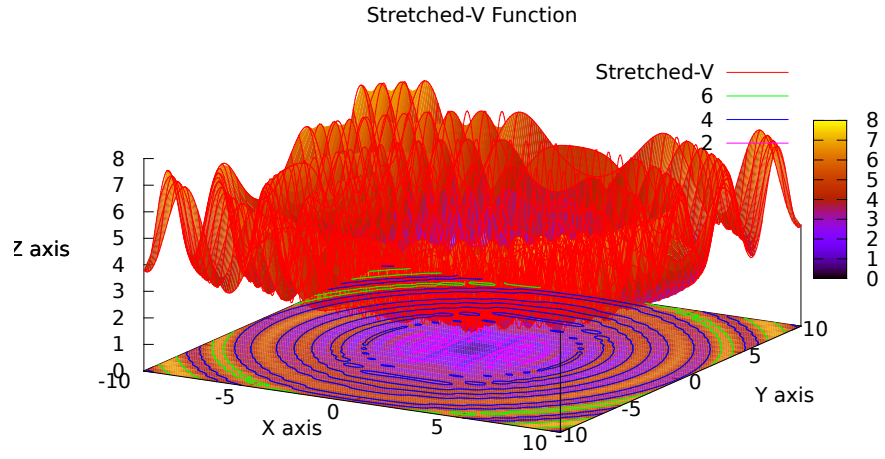


Figure 6.10: Stretched-V Sine Wave Function

6.1.2.6 Michalewics Function

Michalewics function [6.11](#) is a multi-modal function with parameter m which changes the steepness of valleys. Larger m leads to more difficult search [\[8\]](#). This function is defined by equation [\(6.11\)](#).

$$F10(x) = \sum_{i=1}^D \sin(x_i) \sin^{2m} \left(\frac{ix_i^2}{\pi} \right); \quad m = 10 \quad (6.11)$$

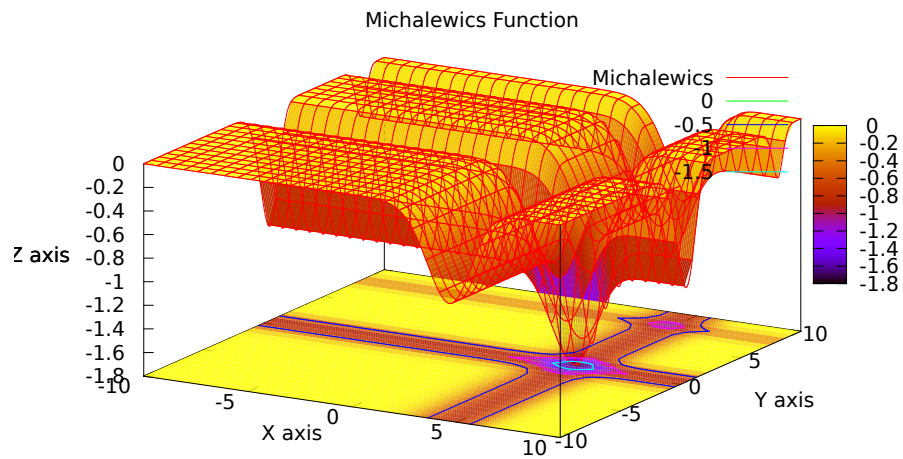


Figure 6.11: Michalewics Function

6.1.3 Binary Only Benchmark Problems

6.1.3.1 DF3 Function

For GA and other optimization algorithms as well, the most challenging problems are deceptive. And one such problem is "deceptive function 3" [13]. This function is designed for binary strings only and can't be used with real parameters. To evaluate a binary string, it must be first divided into four bit long parts. Each part is then evaluated by table 6.1 and all evaluations are added together.

$f(1111) = 0$	$f(0100) = 3$	$f(0110) = 25$	$f(1110) = 30$
$f(0000) = 20$	$f(1000) = 2$	$f(1001) = 25$	$f(1101) = 30$
$f(0001) = 5$	$f(0011) = 25$	$f(1010) = 25$	$f(1011) = 30$
$f(0010) = 4$	$f(0101) = 25$	$f(1100) = 25$	$f(0111) = 30$

Table 6.1: Values for "deceptive function 3"

6.2 Experimental Configuration

With PyPGA package, several experiments we implemented. There are three heterogeneous experiments and for each different island in them, one homogeneous experiment was executed so that results can be compared.

All experiments have common following parameters:

- There are five islands in the *World*.
- Each island contains 50 individuals.
- All problems are solved in 5D.
- Search space is constrained into $< -10, 10 >$ interval.
- 50 children are generated each generation.
- The experiment runs for 400 generations.
- The number of individuals to be accepted with each migration is 5.

6.2.1 Experiments with Float Representation

All experiments described in this section use crossover operators with real chromosome representation 2.2.3.

6.2.1.1 PGA-1-real

The first experiment with real coded chromosomes has five identical islands with following setting:

- CX operator is set to mean crossover (see section 2.4.1).
- Parent selection is set to greedy over-selection (see section 2.7.3).
- Evolution model is set to generational model (see section 2.6.1).
- Migration period is set to 10 generations.
- Probability of mutation is set to 10%.

6.2.1.2 PGA-2-real

The second experiment with real coded chromosomes has five identical islands with following setting:

- CX operator is set to $BLX - \alpha$ (see section 2.4.2).
- Parent selection is set to greedy over-selection (see section 2.7.3).
- Evolution model is set to generational model (see section 2.6.1).
- Migration period is set to 10 generations.
- Probability of mutation is set to 10%.

6.2.1.3 PGA-3-real

The third experiment with real coded chromosomes has five identical islands with following setting:

- CX operator is set to SBX (see section 2.4.3).
- Parent selection is set to tournament selection (see section 2.7.1).
- Evolution model is set to generational model (see section 2.6.1).
- Migration period is set to 10 generations.
- Probability of mutation is set to 10%.

6.2.1.4 PGA-4-real

The fourth experiment with real coded chromosomes has five identical islands with following setting:

- CX operator is set to SBX (see section 2.4.3).
- Parent selection is set to tournament selection (see section 2.7.1).
- Evolution model is set to steady state model (see section 2.6.2).
- Migration period is set to 10 generations.
- Probability of mutation is set to 10%.

6.2.1.5 PGA-5-real

The fifth experiment with real coded chromosomes has five identical islands with following setting:

- CX operator is set to SBX (see section 2.4.3).
- Parent selection is set to truncation selection (see section 2.7.4).
- Evolution model is set to generational model (see section 2.6.1).
- Migration period is set to 80 generations.
- Probability of mutation is set to 20%.
- Mutation operator is modified so that it places new values 100000 times further from the original value than normally.

6.2.1.6 PGA-mix-real

The last experiment with real coded chromosomes is a combination of all previous experiments, one island from each. This experiment is the only heterogeneous experiment in this set.

6.2.2 Experiments with Direct Representation

All experiments described in this section use crossover operators with direct chromosome representation (see section 2.2.1). Those experiments have following common setting:

- Chance for crossover is set to 90%.
- Chance for mutation is set to 10%.
- Migration period is set to 10 generations.
- The number of individuals to be accepted with each migration is 5.

6.2.2.1 PGA-1-bin

The first experiment with direct coded chromosomes has five identical islands with following setting:

- CX operator is set to uniform crossover (see section [2.3.3](#)).
- Parent selection is set to tournament selection (see section [2.7.1](#)).
- Evolution model is set to steady state model (see section [2.6.2](#)).
- Probability of crossover is set to 90%.
- Probability of mutation is set to 10%.

6.2.2.2 PGA-2-bin

The second experiment with direct coded chromosomes has five identical islands with following setting:

- CX operator is set to two-point crossover (see section [2.3.2](#)).
- Parent selection is set to tournament selection (see section [2.7.1](#)).
- Evolution model is set to generational model (see section [2.6.1](#)).
- Probability of crossover is set to 90%.
- Probability of mutation is set to 10%.

6.2.2.3 PGA-3-bin

The third experiment with direct coded chromosomes has five identical islands with following setting:

- CX operator is set to two-point crossover (see section [2.3.2](#)).
- Parent selection is set to tournament selection (see section [2.7.1](#)).
- Evolution model is set to steady state model (see section [2.6.2](#)).
- Probability of crossover is set to 90%.
- Probability of mutation is set to 10%.

6.2.2.4 PGA-4-bin

The fourth experiment with direct coded chromosomes has five identical islands with following setting:

- CX operator is set to uniform crossover (see section 2.3.3).
- Parent selection is set to tournament selection (see section 2.7.1).
- Evolution model is set to generational model (see section 2.6.1).
- Probability of crossover is set to 90%.
- Probability of mutation is set to 10%.

6.2.2.5 PGA-5-bin

The fifth experiment with direct coded chromosomes has five identical islands with following setting:

- CX operator is set to two-point crossover (see section 2.3.3).
- Parent selection is set to tournament selection (see section 2.7.1).
- Evolution model is set to generational model (see section 2.6.1).
- Probability of crossover is set to 70%.
- Probability of mutation is set to 40%.
- When mutation occurs 10 random bits are changed.

6.2.2.6 PGA-mix-bin

The last experiment with direct coded chromosomes is a combination of all previous experiments, one island from each. This experiment is the only heterogeneous experiment in this set.

6.2.3 Experiment with DF3 Function

All experiments described in this section have deceptive function 3 (see section 6.1.3.1) set as their fitness function, since df3 takes binary string as an argument there is no point in distinguishing different representations. The length of the binary string for this experiment is set to 100 bits. Following setting is common to all experiments:

- Parent selection is set to tournament (see section 2.7.1).
- Migration period is set to 10 generations.
- Probability of mutation is set to 15%.
- Probability of crossover is set to 95%.
-

6.2.3.1 PGA-1-df3

The first experiment with df3 function has five identical islands with following setting:

- CX operator is set to two point crossover (see section 2.3.2).
- Evolution model is set to generational model (see section 2.6.1).

6.2.3.2 PGA-2-df3

The second experiment with df3 function has five identical islands with following setting:

- CX operator is set to uniform crossover (see section 2.3.3).
- Evolution model is set to generational model (see section 2.6.1).

6.2.3.3 PGA-3-df3

The third experiment with df3 function has five identical islands with following setting:

- CX operator is set to two point crossover (see section 2.3.2).
- Evolution model is set to steady state model (see section 2.6.2).

6.2.3.4 PGA-4-df3

The fourth experiment with df3 function has five identical islands with following setting:

- CX operator is set to uniform crossover (see section 2.3.3).
- Evolution model is set to steady state model (see section 2.6.2).

6.2.3.5 PGA-5-df3

The fifth experiment with df3 function has five identical islands with following setting:

- CX operator is set to two-point crossover (see section 2.3.3).
- Parent selection is set to tournament selection (see section 2.7.1).
- Evolution model is set to generational model (see section 2.6.1).
- Probability of crossover is set to 70%.
- Probability of mutation is set to 40%.
- When mutation occurs 10 random bits are changed.

6.2.3.6 PGA-mix-df3

The last experiment with df3 function is a combination of all previous experiment, one island from each. This experiment is the only heterogeneous experiment in this set.

6.3 Experimental Results

All configuration described in section 6.2 have been tested with functions described in chapter 6.1. Every experiment was executed thirty times and the best achieved fitness was collected from each run. Results for each homogeneous configuration were then compared with those from the heterogeneous configuration. To determine if those results are statistically significant Wilcoxon rank-sum test is used.

6.3.1 Experiments with Real Coded Chromosomes

The median value value from all 30 runs of every experiment is in table 6.2. It can be seen from table of ρ -statistic 6.4 that heterogeneous PGA shows better results for all but three experiments. When table of P-values 6.3 is consulted, it can be seen that only one of those results is statistically significant.

For sphere function, ellipsoidal function and quartic function, heterogeneous configuration was better than any homogeneous in every run, as can be seen from table 6.2 there was a difference of several orders.

6.3.2 Experiments with Direct Coded Chromosomes

The results for experiments with binary encoded chromosomes are comparable to those with real coded chromosomes. The main differences are in the ellipsoidal, sphere and quartic functions. Since this representation has limited precision, most GAs were able to find the best possible solution in every run.

For the rest of those functions, heterogeneous PGA shows satisfactory results. As can be seen from tables 6.6 and 6.7, it has similar or better results than experiments PGA-1-bin up to PGA-4-bin. When compared to PGA-5-bin, heterogeneous model is better in three cases and worse also in three cases.

6.3.3 df3 Experiment

The results for experiments with df3 function are shown in table 6.8. Heterogeneous model did not accomplish any significant improvement against homogeneous models, in fact its results are average. It did much better than experiments with steady state evolution model but it failed to exceed those with generational evolution model.

Test Function	PGA-1-real	PGA-2-real	PGA-3-real
Rastrigin	2.68642329692	0.0314731909984	4.97479024765
Schwefel	685.154048135	$7.77153445597 \cdot 10^{-05}$	118.43833465
sphere	0.00825410263286	$7.97414742669 \cdot 10^{-09}$	$4.59549607292 \cdot 10^{-26}$
Rosenbrock	4.47861364706	0.60143126212	1.56684139726
Michalewics	-3.7348528679	-4.6876268184	-4.49478058381
pathological	0.0124442964861	0.121741453869	0.274540237362
quartic	$6.44630114453 \cdot 10^{-05}$	$2.20430733108 \cdot 10^{-15}$	$3.04780222547 \cdot 10^{-26}$
ellipsoidal	0.03031079433	$1.44044664424 \cdot 10^{-08}$	$3.51598305269 \cdot 10^{-24}$
stretched-V	0.0038751122757	0.000153102881627	$8.57097397757 \cdot 10^{-11}$
Schwefel's sum	0.0117300003474	$2.33717638449 \cdot 10^{-09}$	$1.06429869037 \cdot 10^{-05}$
masters	-3.70177529513	-3.70388180863	-3.70755960932
Test Function	PGA-4-real	PGA-5-real	PGA-mix-real
Rastrigin	7.95967891917	$1.42108547152 \cdot 10^{-14}$	0.994959057093
Schwefel	236.877574813	$9.09494701773 \cdot 10^{-13}$	$4.54747350886 \cdot 10^{-13}$
sphere	$5.5651170302 \cdot 10^{-09}$	$1.13164141549 \cdot 10^{-41}$	$9.59990873884 \cdot 10^{-148}$
Rosenbrock	3.2522858391	0.909349255918	0.626348905181
Michalewics	-4.49589320653	-4.68765817909	-4.64589536775
pathological	0.283920555527	0.0351711848968	0.032720947617
quartic	$8.08750994685 \cdot 10^{-11}$	$3.04013160864 \cdot 10^{-60}$	$1.27847305488 \cdot 10^{-205}$
ellipsoidal	$1.1680184969 \cdot 10^{-10}$	$3.41504250838 \cdot 10^{-41}$	$7.66556954899 \cdot 10^{-146}$
stretched-V	$1.18811889704 \cdot 10^{-05}$	0.0	0.0
Schwefel's sum	0.00155101043485	$4.54483150799 \cdot 10^{-06}$	$9.17009025845 \cdot 10^{-13}$
masters	-3.28560911365	-3.70755960932	-3.70755960932

Table 6.2: Median fitness values for experiments with real coded chromosomes

Test Function	PGA-1-real	PGA-2-real	PGA-3-real
Rastrigin	$8.12107314729 \cdot 10^{-09}$	0.468797586246	$3.50974804775 \cdot 10^{-11}$
Schwefel	$3.17521564597 \cdot 10^{-11}$	0.12057499226	0.0159585842613
sphere	$2.87194712456 \cdot 10^{-11}$	$2.87194712456 \cdot 10^{-11}$	$2.87194712456 \cdot 10^{-11}$
Rosenbrock	$7.76172459638 \cdot 10^{-11}$	0.594560256544	0.00496893443565
Michalewics	$2.0532686662 \cdot 10^{-10}$	0.722719791571	$1.79377664811 \cdot 10^{-06}$
pathological	0.00341901044855	$1.14992747346 \cdot 10^{-06}$	0.000232027814361
quartic	$2.87194712456 \cdot 10^{-11}$	$2.87194712456 \cdot 10^{-11}$	$2.87194712456 \cdot 10^{-11}$
ellipsoidal	$2.87194712456 \cdot 10^{-11}$	$2.87194712456 \cdot 10^{-11}$	$2.87194712456 \cdot 10^{-11}$
stretched-V	$1.04155128966 \cdot 10^{-10}$	$8.48722425673 \cdot 10^{-10}$	0.117079846005
Schwefel's sum	$2.87194712456 \cdot 10^{-11}$	$8.0169641592 \cdot 10^{-08}$	$5.77298209237 \cdot 10^{-11}$
masters	$2.87194712456 \cdot 10^{-11}$	$2.87194712456 \cdot 10^{-11}$	0.836023950489
Test Function	PGA-4-real	PGA-5-real	
Rastrigin	$5.77298209237 \cdot 10^{-11}$	0.164609061949	
Schwefel	$2.29022487641 \cdot 10^{-08}$	0.534635240704	
sphere	$2.87194712456 \cdot 10^{-11}$	$2.87194712456 \cdot 10^{-11}$	
Rosenbrock	$4.61923248984 \cdot 10^{-07}$	0.0443590716039	
Michalewics	$9.89854166855 \cdot 10^{-07}$	0.767467973961	
pathological	$9.18087424306 \cdot 10^{-07}$	0.689760957276	
quartic	$2.87194712456 \cdot 10^{-11}$	$2.87194712456 \cdot 10^{-11}$	
ellipsoidal	$2.87194712456 \cdot 10^{-11}$	$2.87194712456 \cdot 10^{-11}$	
stretched-V	$1.77392789347 \cdot 10^{-09}$	0.198357566663	
Schwefel's sum	$2.87194712456 \cdot 10^{-11}$	$5.22830667649 \cdot 10^{-11}$	
masters	$2.87194712456 \cdot 10^{-11}$	$4.13418361767 \cdot 10^{-08}$	

Table 6.3: P-values for experiments with real coded chromosomes

Test Function	PGA-1-real	PGA-2-real	PGA-3-real	PGA-4-real	PGA-5-real
Rastrigin	0.9333	0.5544	1.0	0.9922	0.55
Schwefel	0.9989	0.6167	0.8411	0.92	0.68
sphere	1.0	1.0	1.0	1.0	1.0
Rosenbrock	0.9889	0.46	0.7111	0.8789	0.6511
Michalewics	0.9778	0.5267	0.8589	0.8678	0.4344
pathological	0.28	0.8656	0.7767	0.8689	0.53
quartic	1.0	1.0	1.0	1.0	1.0
ellipsoidal	1.0	1.0	1.0	1.0	1.0
stretched-V	0.9856	0.9611	0.7756	0.9522	0.6522
Schwefel's sum	1.0	0.9033	0.9922	1.0	0.9933
masters	1.0	1.0	0.8578	1.0	0.7511

Table 6.4: ρ -statistic values for experiments with real coded chromosomes

Test Function	PGA-1-bin	PGA-2-bin	PGA-3-bin
Rastrigin	8.34120484702	3.07017159868	4.97582767759
Schwefel	34.6546348771	0.311601326484	0.312226411768
sphere	$4.54748218249 \cdot 10^{-10}$	$4.54748218249 \cdot 10^{-10}$	$4.54748218249 \cdot 10^{-10}$
Rosenbrock	4.59786153483	3.99015946567	4.43120043886
Michalewics	-4.58607133613	-4.68494565819	-4.64350072279
pathological	0.00422949060419	0.00156596445197	0.00185783705424
quartic	$1.240775652 \cdot 10^{-19}$	$1.240775652 \cdot 10^{-19}$	$1.240775652 \cdot 10^{-19}$
ellipsoidal	$1.36424465475 \cdot 10^{-09}$	$1.36424465475 \cdot 10^{-09}$	$1.36424465475 \cdot 10^{-09}$
stretched-V	0.00675985242698	0.00136533889282	0.00505994460677
Schwefel's sum	0.195318833551	0.195318833915	0.260556614218
masters	-3.70520686943	-3.7058620205	-3.70590284231
Test Function	PGA-4-bin	PGA-5-bin	PGA-mix-bin
Rastrigin	4.92090766906	0.00123136667715	0.994960888044
Schwefel	0.312851497053	0.212514482884	0.208359310348
sphere	$4.54748218249 \cdot 10^{-10}$	$1.90748687627 \cdot 10^{-06}$	$4.54748218249 \cdot 10^{-10}$
Rosenbrock	4.5365185986	3.75174712094	3.98005561148
Michalewics	-4.63136361624	-4.68622112381	-4.6852626402
pathological	0.00270303083996	0.000582148173629	0.00115716033953
quartic	$1.240775652 \cdot 10^{-19}$	$6.915967935 \cdot 10^{-12}$	$1.240775652 \cdot 10^{-19}$
ellipsoidal	$1.36424465475 \cdot 10^{-09}$	$1.25611462341 \cdot 10^{-05}$	$1.36424465475 \cdot 10^{-09}$
stretched-V	0.00275152644163	$7.11240947651 \cdot 10^{-05}$	0.000779480651169
Schwefel's sum	0.048831198916	$1.3520574025 \cdot 10^{-06}$	$7.56882934455 \cdot 10^{-07}$
masters	-3.6959558452	-3.70740470635	-3.70683022804

Table 6.5: Median fitness values for experiments with direct coded chromosomes

Test Function	PGA-1-bin	PGA-2-bin	PGA-3-bin
Rastrigin	$3.05612801732 \cdot 10^{-09}$	$5.6570273077 \cdot 10^{-06}$	$1.79986793691 \cdot 10^{-07}$
Schwefel	$2.6754924809 \cdot 10^{-07}$	0.0153233302434	0.0172989009604
sphere	1.0	1.0	1.0
Rosenbrock	0.00569773419407	0.63614068577	0.027603250803
Michalewics	$6.97449658915 \cdot 10^{-06}$	0.94107234914	0.0564958741768
pathological	0.00181149797611	0.917573309613	0.468797586246
quartic	1.0	1.0	1.0
ellipsoidal	1.0	1.0	1.0
stretched-V	$2.6836065103 \cdot 10^{-05}$	0.433289795622	0.011958983569
Schwefel's sum	$1.43757908289 \cdot 10^{-06}$	$3.17521564597 \cdot 10^{-11}$	$2.48787435098 \cdot 10^{-10}$
masters	0.000603610841366	0.0760414758302	0.0810611055554
Test Function	PGA-4-bin	PGA-5-bin	
Rastrigin	$1.77392789347 \cdot 10^{-09}$	0.000484605567805	
Schwefel	0.00190453216445	0.0345005081441	
sphere	1.0	$2.87194712456 \cdot 10^{-11}$	
Rosenbrock	0.00967385065092	0.193248378919	
Michalewics	0.00652159522961	0.103888659318	
pathological	0.0187369441809	0.0153233302434	
quartic	1.0	$2.87194712456 \cdot 10^{-11}$	
ellipsoidal	1.0	$2.87194712456 \cdot 10^{-11}$	
stretched-V	0.188236525377	0.00126853453938	
Schwefel's sum	$1.53366757294 \cdot 10^{-07}$	0.554267836324	
masters	0.00393953765506	0.000879453706473	

Table 6.6: P-values for experiments with direct coded chromosomes

Test Function	PGA-1-bin	PGA-2-bin	PGA-3-bin	PGA-4-bin	PGA-5-bin
Rastrigin	0.9456	0.8389	0.8878	0.9522	0.2356
Schwefel	0.9	0.6822	0.7089	0.7333	0.6589
sphere	0.5	0.5	0.5	0.5	1.0
Rosenbrock	0.7078	0.5356	0.6656	0.6944	0.4022
Michalewics	0.8378	0.4944	0.6433	0.7044	0.3778
pathological	0.7344	0.4922	0.5544	0.6767	0.3178
quartic	0.5	0.5	0.5	0.5	1.0
ellipsoidal	0.5	0.5	0.5	0.5	1.0
stretched-V	0.8156	0.5589	0.6889	0.5989	0.2578
schwefel's sum	0.8644	0.9989	0.9756	0.8944	0.5444
masters	0.7578	0.6333	0.6311	0.7167	0.25

Table 6.7: ρ values for experiments with direct coded chromosomes

	PGA-1-df3	PGA-2-df3	PGA-3-df3
mean values	38.8	59.6333333333	41.8333333333
median values	39.0	61.0	42.0
ρ -statistic	0.2256	0.9044	0.4722
P-value	0.000260456253335	$7.385504186 \cdot 10^{-08}$	0.711672625487
	PGA-4-df3	PGA-5-df3	PGA-mix-df3
mean values	62.3333333333	41.4666666667	48.0
median values	62.0	43.0	48.0
ρ -statistic	0.92	0.39	N/A
P-value	$2.29022487641 \cdot 10^{-08}$	0.139289365687	1.0

Table 6.8: Results for experiments with df3 fitness function.

Chapter 7

Conclusion

The main objective of this work was to analyse the performance and advantages of heterogeneous parallel genetic algorithms in comparison to homogeneous ones. Necessary software was developed and several experiments were carried out and while the results are interesting they are also certainly unforeseen. As traditional parallel genetic algorithms compared with single population GAs gives better result for multi-modal problems, it was expected that heterogeneous PGAs will improve this capability even more. But the fact is that heterogeneous PGAs showed improvement in the area of uni-modal testing problems.

This work shows that the use of heterogeneous PGAs is beneficial but there is still much work to be done in this field. Future development can be done with advanced migration topologies specially designed for heterogeneous PGA and dynamic migration control can also bring some improvements.

Bibliography

- [1] J. E. Baker. Reducing bias and inefficiency in the selection algorithm. In *Proceedings of the Second International Conference on Genetic Algorithms on Genetic algorithms and their application*, pages 14–21, Hillsdale, NJ, USA, 1987. L. Erlbaum Associates Inc.
- [2] K. Deb and R. B. Agrawal. Simulated binary crossover for continuous search space. Technical report, Departement of Mechanical Engineering, Indian Institute of Technology, Kanpur, India, 1994.
- [3] K. Deb and H.-g. Beyer. Self-adaptive genetic algorithms with simulated binary crossover. *Evol. Comput.*, 9(2):197–221, 2001.
- [4] D. Goldberg and K. Deb. A comparative analysis of selection schemes used in genetic algorithms. In *Foundations of genetic algorithms (FOGA 1)*, volume 1, pages 69–93, San Francisco, CA, USA, 1991. Morgan Kaufmann.
- [5] IEEE Task P754. *ANSI/IEEE 754-1985, Standard for Binary Floating-Point Arithmetic*. IEEE, New York, NY, USA, Aug. 1985.
- [6] K. D. Jong. *An analysis of the behavior of a class of genetic adaptive systems*. PhD thesis, University of Michigan, Ann Arbor, MI, USA, 1975.
- [7] D. J. Kalyanmoy Deb and A. Anand. Real-coded evolutionary algorithms with parent-centric recombination. *EVOLUTIONARY COMPUTATION*, VOL 10:371–396, 2002.
- [8] C. S. M. Molga. Test functions for optimization needs. www.zsd.ict.pwr.wroc.pl/files/docs/functions.pdf, May 2005.
- [9] D. Ortiz-Boyer, C. Hervás-Martínez, and N. García-Pedrajas. Cixl2: A crossover operator for evolutionary algorithms based on population features. *J. Artif. Intell. Res. (JAIR)*, 24:1–48, 2005.
- [10] M. Pant, M. Ali, and V. P. Singh. Differential evolution with parent centric crossover. In *EMS '08: Proceedings of the 2008 Second UKSIM European Symposium on Computer Modeling and Simulation*, pages 141–146, Washington, DC, USA, 2008. IEEE Computer Society.
- [11] M. Rucinski, D. Izzo, and F. Biscani. On the impact of the migration topology on the island model. *Parallel Computing*, In Press, Corrected Proof, 2010.

- [12] J. Wakunda and A. Zell. Median-selection for parallel steady-state evolution strategies. In M. Schoenauer, K. Deb, G. Rudolph, X. Yao, E. Lutton, J. J. M. Guervós, and H.-P. Schwefel, editors, *PPSN*, volume 1917 of *Lecture Notes in Computer Science*, pages 405–414. Springer, 2000.
- [13] L. D. Whitley. Fundamental principles of deception in genetic search. In G. J. E. Rawlins, editor, *FOGA*, pages 221–241. Morgan Kaufmann, 1990.
- [14] L. D. Whitley. A genetic algorithm tutorial. *Statistics and Computing*, 4:65–85, 1994.
- [15] A. H. Wright. Genetic algorithms for real parameter optimization. In G. J. Rawlins, editor, *Foundations of genetic algorithms*, pages 205–218. Morgan Kaufmann, San Mateo, CA, 1991.

Appendix A

List of Abbreviations

2D	Two-Dimensional
5D	Five-Dimensional
BLX-α	Blend Crossover
CX	Crossover
df3	Fully deceptive function 3
EA	Evolution Algorithm
g3	Generalized Generation Gap
GA	Genetic Algorithm
IM	Island Model
MGG	Minimal Generation Gap Model
PCX	Parent Centric Crossover
PGA	Parallel Genetic Algorithm
SBX	Simulated Binary Crossover
SUS	Stochastic Universal Sampling
UNDX	Uni-modal Normal Distribution Crossover

Appendix B

The Wilcoxon Ran-Sum Test

To determine the statistical significance of test results *Wilcoxon rank-sum test* (also called *Mann-Whitney U*) has been used. It is a non parametric test used to estimate whether two samples are independent or not.

B.1 Calculation

To calculate the value of statistic **U** following method can be used:

1. Arrange all the observations into single ranked series.
2. Sum all ranks R_1 for the observations which came from the first series.
3. Calculate U_1 using equation (B.1) where n_1 is the size of the series.
4. Value of U_2 can be calculated using equation (B.2).

$$U_1 = R_1 \frac{n_1(n_1 + 1)}{2} \quad (\text{B.1})$$

$$U_2 = R_2 \frac{n_2(n_2 + 1)}{2} \quad (\text{B.2})$$

From the equations (B.1) and (B.2) can be deduced the maximal value of U_1 which is the sum of $U_1 + U_2 = n_1 n_2$ when $U_2 = 0$.

B.2 P-value

P-value in this test represents the probability that two series of observations came from the same set. In other words, if the P-value is large, then both series have similar distribution and small P-value means that both series are most probably independent.

B.3 ρ Statistic

ρ statistic is calculated by dividing U by its maximal value n_1n_2 , thus it is linearly related to U . It is a non-parametric measure of overlap between two distributions, it can take values from the interval $< 0, 1 >$ with both boundaries meaning no overlap. It can also be used to say which distribution has better results even when medians are the same.

Appendix C

CD Content

```
|-- app/
|   |-- example.py           - Example experiment
|   |-- experiment_binary.py - Experiment with Direct coded chromosomes
|   |-- experiment_df3.py    - df3 experiment
|   |-- experiment_float.py  - Experiment with real coded chromosomes
|   |-- PyPGA/              - PyPGA package
|       |-- fitness.py
|       |-- genome.py
|       |-- __init__.py
|       |-- organism.py
|       |-- population.py
|       |-- util.py
|       |-- world.py
|   |-- util/
|       |-- ranksum.py      - Script used to process results with Rank-Sum test
|-- python/
|   |-- numpy-1.4.1.tar.gz   - NumPy python package
|   |-- python-3.1.2.amd64.msi - Python binary for MS Windows
|   |-- python-3.1.2-macosx10.3-2010-03-24.dmg - Python binary for MacOS X
|   |-- python-3.1.2.msi    - Python binary for MS Windows (32bit)
|   |-- Python-3.1.2.tar.bz2 - Python sources
|   |-- scipy-0.7.2.tar.gz  - SciPy python package
|-- text/
|   |-- Cerny-2010.pdf      - This text in pdf format
|--README.txt              - README file
```