TUGAS AKHIR - KM184801

# ALGORITMA GENETIKA UNTUK OPTIMASI FUNGSI DENGAN BANYAK VARIABEL

**FITYAN AZIZI**
**06111640000103**

Dosen Pembimbing
Drs. Bandung Arry S., MI.Komp
Muhammad Luthfi S., S.Si, M.Si

Departemen Matematika
Fakultas Sains dan Analitika Data
Institut Teknologi Sepuluh Nopember
Surabaya 2020

TUGAS AKHIR - KM184801

# ALGORITMA GENETIKA UNTUK OPTIMASI FUNGSI DENGAN BANYAK VARIABEL

FITYAN AZIZI
NRP 06111640000103

Dosen Pembimbing:
Drs. Bandung Arry S., MI.Komp
Muhammad Luthfi Shahab, S.Si, M.Si

DEPARTEMEN MATEMATIKA
Fakultas Sains dan Analitika Data
Institut Teknologi Sepuluh Nopember
Surabaya 2020

FINAL PROJECT - KM4801

# A GENETIC ALGORITHM FOR MULTIVARIABLE FUNCTION OPTIMIZATION

FITYAN AZIZI
NRP 06111640000103

Supervisors:
Drs. Bandung Arry S., MI.Komp
Muhammad Luthfi Shahab, S.Si, M.Si

DEPARTMENT OF MATHEMATICS
Faculty of Science and Data Analytics
Institut Teknologi Sepuluh Nopember
Surabaya 2020

LEMBAR PENGESAHAN

ALGORITMA GENETIKA UNTUK OPTIMASI FUNGSI DENGAN BANYAK VARIABEL

*A GENETIC ALGORITHM FOR MULTIVARIABLE FUNCTION OPTIMIZATION*

Diajukan Untuk Memenuhi Salah Satu Syarat
Untuk Memperoleh Gelar Sarjana Matematika
Pada bidang studi Ilmu Komputer
Program Studi S-1 Departemen Matematika
Fakultas Sains dan Analitika Data
Institut Teknologi Sepuluh Nopember Surabaya

Oleh:
FITYAN AZIZI
NRP. 06111640000103

Menyetujui,

| Dosen Pembimbing II, | Dosen Pembimbing I, |
|---|---|
| M. Luthfi Shahab, S.Si, M.Si | Drs. Bandung Arry S., MI.Komp |
| NIP. 19950331 201803 1 001 | NIP. 19630605 198903 1 003 |

Mengetahui,
Kepala Departemen Matematika
FSAD ITS

Subchan, Ph.D
NIP. 19710513 199702 1 001

Surabaya, Agustus 2020



Scanned with CamScanner

# ALGORITMA GENETIKA UNTUK OPTIMASI FUNGSI DENGAN BANYAK VARIABEL


| | | |
|---|---|---|
| Nama Mahasiswa | : | Fityan Azizi |
| NRP | : | 06111640000103 |
| Jurusan | : | Matematika FSAD-ITS |
| Pembimbing | : | 1. Drs. Bandung Arry S., MI.Komp |
| | | 2. Muhammad Luthfi Shahab, S.Si, M.Si |



**Abstrak**

*Dalam berbagai aplikasi pada kehidupan, banyak masalah optimasi yang memiliki banyak variabel. Salah satu algoritma yang dapat digunakan untuk menyelesaikan masalah optimasi fungsi adalah algoritma genetika. Namun, masih sedikit penelitian tentang penggunaan algoritma genetika yang mampu mengoptimasi fungsi dengan banyak variabel, misalnya hingga 100000 variabel. Dalam penelitian ini, dirumuskan algoritma genetika yang mampu mengoptimasi fungsi dengan banyak variabel. Dalam algoritma genetika yang telah dirumuskan, perhitungan nilai fitness dilakukan ketika proses crossover dan mutasi, sehingga setelah mendapatkan kromosom baru, tidak perlu dilakukan perhitungan untuk mencari nilai fitness. Algoritma genetika yang sudah dirumuskan kemudian diuji dengan enam fungsi dengan karakteristik yang berbeda. Fungsi tersebut adalah fungsi sphere, csendes, alpine, sum squares, schumer-steiglitz, dan quintic. Hasilnya, algoritma genetika yang telah dirumuskan mampu menghasilkan solusi yang baik, khususnya pada fungsi sphere, csendes, sum squares, schumer-stiglitz, dan quintic.*



***Kata kunci** : Optimasi fungsi, algoritma genetika, variabel*

# A GENETIC ALGORITHM FOR MULTIVARIABLE FUNCTION OPTIMIZATION


| | | |
|---|---|---|
| Name | : | Fityan Azizi |
| NRP | : | 06111640000103 |
| Department | : | Mathematics FSDA-ITS |
| Supervisors | : | 1. Drs. Bandung Arry S., MI.Komp |
| | | 2. Muhammad Luthfi Shahab, S.Si, M.Si |



**Abstract**

*In various applications in life, many optimization problems that have many variables, One algorithm that can be used to solve the optimization problem is genetic algorithm. However, there is still few research on the use of genetic algorithms that are able to optimize functions with many variables, for example up to 100000 variables. In the genetic algorithm that has been formulated, the calculation of the fitness value is done during the crossover and mutation process, so that after getting a new chromosome, there is no need to do the calculation to find the fitness value. The genetic algorithm that has been formulated is then tested with six functions with different characteristics. These functions are sphere, csendes, alpine, sum squares, schumer-steiglitz, and quintic functions. As a result, genetic algorithms that have been formulated are capable of producing good solutions, especially in the sphere, csendes, sum squares, schumer-stiglitz, and quintic functions.*



***Keywords*** *: Function optimization, genetic algorithm, variable*

# KATA PENGANTAR

Alhamdulillahirobbil'alamin. Puji syukur penulis panjatkan atas kehadirat Allah SWT, yang telah memberikan berbagai nikmat sehingga penulis dapat menyelesaikan tugas akhir berjudul **"Algoritma Genetika untuk Optimasi Fungsi dengan Banyak Variabel"**. Sholawat serta salam kepada nabi Muhammad SAW, kepada keluarganya, para sahabat, dan pengikutnya.

Ucapan terimakasih kepada pihak-pihak yang telah membantu penulis dalam menyelesaikan penelitian ini. Khususnya:

1. Kedua orangtua penulis, Teguh Dwi Nugroho, S.E, M.M. dan Nyayu Talha atas segala dukungan dalam mengerjakan dan menyelesaikan tugas akhir.

2. Bapak Drs. Bandung Arry S., MI.Komp., Selaku dosen pembimbing pertama yang telah memberikan bimbingan, arahan, dan waktu yang diberikan kepada penulis.

3. Bapak Muhammad Luthfi Shahab, S.Si, M.Si., Selaku dosen pembimbing kedua yang juga telah memberikan bimbingan, arahan, dan waktu yang diberikan kepada penulis.

4. Bapak Subchan, Ph.D, selaku Kepala Departemen Matematika Institut Teknologi Sepuluh Nopember Surabaya.

5. Bapak Dr. Imam Mukhlash, S.Si., M.T., Bapak Drs. Kamiran, M.Si., dan Ibu Alvida Mustika Rukmi, S.Si, M.Si. Selaku dosen penguji yang telah memberikan saran dan arahan untuk tugas akhir ini.

6. Bapak Dr. Didik Khusnul Arif, S.Si, M.Si. dan Ibu Alvida Mustika Rukmi, S.Si, M.Si., selaku dosen wali yang telah memberikan bimbingan selama penulis menempuh perkuliahan di Departemen Matematika ITS.

7. Bapak dan Ibu dosen Departemen Matematika ITS, atas ilmu yang telah diberikan selama penulis menempuh perkuliahan di Departemen Matematika ITS.

8. Kawan-kawan "Obrolan Hangat BC", semoga obrolan kita selalu "Hangat" sampai kapanpun.

Penulis sadar bahwa Tugas Akhir ini masih jauh dari kesempurnaan. Oleh karena itu, penulis mengharapkan kritik dan saran demi kesempurnaan Tugas Akhir ini. Program dalam Tugas Akhir yang telah disusun diimplementasikan dalam laman projectgenetic.herokuapp.com. Semoga Tugas Akhir ini dapat bermanfaat bagi pihak yang berkepentingan.

Surabaya, Agustus 2020

Penulis

# DAFTAR ISI


HALAMAN JUDUL i

LEMBAR PENGESAHAN v

ABSTRAK vii

ABSTRACT ix

KATA PENGANTAR xi

DAFTAR ISI xiii

DAFTAR GAMBAR xvii

DAFTAR TABEL xix

DAFTAR LAMPIRAN xxi

BAB I PENDAHULUAN 1

1.1 Latar Belakang . . . . . 1

1.2 Rumusan Masalah . . . . . 4

1.3 Batasan Masalah . . . . . 4

1.4 Tujuan . . . . . 4

1.5 Manfaat . . . . . 5

BAB II TINJAUAN PUSTAKA 7

2.1 Penelitian Terdahulu . . . . . 7

2.2 Algoritma Genetika . . . . . 7

2.2.1 Tournament Selection . . . . . 8

2.2.2 Average Crossover . . . . . 10

2.2.3 Desain Awal Kromosom . . . . . 10

2.3 Algoritma penjumlahan Neumaier . . . . . 10

2.4 Fungsi . . . . . 11

BAB III METODE PENELITIAN 15
3.1 Studi literatur . . . . . 15
3.2 Mencari data . . . . . 15
3.3 Perumusan Algoritma Genetika untuk optimasi fungsi dengan banyak Variabel . . . . 15
3.4 Perancangan program . . . . . 15
3.5 Menghitung Akurasi . . . . . 16
3.6 Penarikan Kesimpulan . . . . . 16

BAB IV ANALISIS DAN PEMBAHASAN 17
4.1 Perumusan Algoritma Genetika untuk Optimasi Fungsi dengan Banyak Variabel . . . 17
4.2 Implementasi Pada Program . . . . . 25
4.3 Akurasi hasil Perumusan Algoritma Genetika untuk Optimasi Fungsi dengan Banyak Variabel . . . . . 29
4.3.1 Uji fungsi Sphere . . . . . 30
4.3.2 Uji fungsi Csendes . . . . . 32
4.3.3 Uji fungsi Alpine . . . . . 34
4.3.4 Uji fungsi Sum Squares . . . . . 35
4.3.5 Uji fungsi Schumer-steiglitz . . . . . 38
4.3.6 Uji fungsi Quintic . . . . . 40
4.4 Analisis Pengaruh Besar Populasi dan Besar Peluang Mutasi Terhadap Hasil Uji . . . . . 42
4.4.1 Uji fungsi Sphere . . . . . 42
4.4.2 Uji fungsi Csendes . . . . . 43
4.4.3 Uji fungsi Alpine . . . . . 45
4.4.4 Uji fungsi Sum Squares . . . . . 46
4.4.5 Uji fungsi Schumer-Steiglitz . . . . . 46
4.4.6 Uji fungsi Quintic . . . . . 47
4.4.7 Analisis Kasus Tidak Berpengaruhnya Perbedaan Besar Populasi Pada Uji Fungsi Tertentu . . . . . 49

BAB V PENUTUP 53
5.1 Kesimpulan . . . . . 53
5.2 Saran . . . . . 54
DAFTAR PUSTAKA 55
LAMPIRAN 59
BIODATA PENULIS 75

# DAFTAR GAMBAR


Gambar 2.1 Langkah-langkah Algoritma Genetika . . 9
Gambar 2.2 Contoh *Average Crossover* . . . . . 10

Gambar 4.1 Langkah-langkah Algoritma Genetika untuk Optimasi Fungsi dengan Banyak Variabel . . . . . 25
Gambar 4.2 Grafik nilai $fitness$ terhadap besar populasi pada uji fungsi *Sphere* . . . . . 43
Gambar 4.3 Grafik nilai $fitness$ terhadap besar populasi pada uji fungsi $Csendes$ . . . . . 44
Gambar 4.4 Grafik nilai $fitness$ terhadap besar populasi pada uji fungsi *Alpine* . . . . . 45
Gambar 4.5 Grafik nilai $fitness$ terhadap besar populasi pada uji fungsi $SumSquares$ . 47
Gambar 4.6 Grafik nilai $fitness$ terhadap besar populasi pada uji fungsi $Schumer - Steiglitz$ . . . . . 48
Gambar 4.7 Grafik nilai $fitness$ terhadap besar populasi pada uji fungsi *Quintic* . . . . . 49

# DAFTAR TABEL

Tabel 4.1 Hasil uji pada fungsi sphere . . . . . 30
Tabel 4.2 Hasil uji pada fungsi csendes . . . . . 32
Tabel 4.3 Hasil uji pada fungsi alpine . . . . . 34
Tabel 4.4 Hasil uji pada fungsi sum squares . . . . . 35
Tabel 4.5 Hasil uji pada fungsi schumer-steiglitz . . . 38
Tabel 4.6 Hasil uji pada fungsi quintic . . . . . 40

## DAFTAR LAMPIRAN

LAMPIRAN 1 *Source code* Library.java . . . . . 61
LAMPIRAN 2 *Source code* Solution.java . . . . . 62
LAMPIRAN 3 *Source code* Population.java . . . . . 67
LAMPIRAN 4 *Source code* GeneticAlgorithm.java . . . 69
LAMPIRAN 5 *Source code* Main.java . . . . . 73

# BAB I
# PENDAHULUAN

Pada bab ini akan dijelaskan tentang latar belakang, rumusan masalah, batasan masalah, tujuan penelitian, dan manfaat penelitian.

## 1.1 Latar Belakang

Optimasi sangat penting dalam berbagai aplikasi, seperti teknik, kegiatan bisnis, dan desain industri. Tujuan optimasi bisa berupa apa saja, untuk meminimalkan biaya dan konsumsi energi, untuk memaksimalkan laba, output, performa, dan efisiensi. Sehingga tidak berlebihan jika mengatakan bahwa optimasi dibutuhkan dimana-mana [1]. Banyak aplikasi optimasi dalam kehidupan yang memiliki banyak variabel. Sebagai contoh, dalam sains data, *data scientist* menggunakan dimensi untuk mendeskripsikan atribut dari *big data* [2]. Dalam aplikasi manufaktur, dihitung minimum biaya yang dikeluarkan untuk produktifitas dari tiap-tiap mesin. Dalam aplikasi maskapai penerbangan, dihitung maksimum keuntungan yang akan didapat dari tiap pesawat dan rute yang berbeda [3]. Tentunya variabel yang ada dalam aplikasi-aplikasi tersebut tidak sedikit. Akan sangat sulit dan membutuhkan waktu yang cukup lama jika dihitung secara manual.

Dalam menyelesaikan masalah optimasi fungsi secara eksak, dibutuhkan formulasi matematika yang kompleks untuk memberikan solusi yang pasti. Solusi optimum membutuhkan proses perhitungan yang panjang dan tidak praktis[4]. Selain dengan cara eksak, ada beberapa metode

yang bisa digunakan dalam menyelesaikan masalah optimasi fungsi. Dalam pemrograman linier, ada tiga metode yang bisa digunakan. yaitu metode aljabar, metode grafik, dan metode simpleks. metode aljabar merupakan teknik yang paling sederhana tetapi kurang efisien, terutama apabila jumlah batasan cukup banyak. Metode grafik juga merupakan cara yang cukup sederhana namun hanya dapat digunakan untuk permasalahan dua variabel, karena jika grafiknya lebih dari dua variabel maka dapat dibayangkan kesulitan yang dialami dalam mencari titik penyelesaian yang optimal. Metode yang populer dalam pemrograman linier adalah metode simpleks. metode simpleks mampu menyelesaikan masalah optimasi dengan banyak variabel dan juga banyak kendala[6]. metode lain yang bisa digunakan dalam optimasi adalah menggunakan metode heuristik[5]. Dalam menyelesaikan masalah optimasi dengan banyak variabel dengan menggunakan metode heuristik, James dan Li membuat sebuah algoritma yang disebut *Social Spider algorithm.* hasilnya algoritma tersebut mampu menyelesaikan masalah optimasi dengan baik dan menggunakan uji berbagai fungsi dengan maksimal variabel mencapai 30 variabel[7]. Selanjutnya, Sun dkk. membuat sebuah algoritma yang disebut *A modified whale optimization algorithm*, algoritma tersebut juga mampu menyelesaikan masalah optimasi fungsi, kali ini algoritma tersebut diuji dengan fungsi yang memiliki variabel mencapai 1000. Dalam penelitian ini, penulis akan menggunakan metode heuristik untuk menyelesaikan masalah optimasi dengan banyak variabel, yaitu menggunakan algoritma genetika.

Dalam menyelesaikan masalah optimasi fungsi menggunakan algoritma genetika, Mahmudy menggunakan algoritma genetika Dengan Kromosom Biner dan Perbaikan Kromosom Hill-Climbing[9]. hasilnya algoritma genetika

tersebut menghasilkan hasil yang baik jika digunakan pada fungsi yang kompleks. Namun, dalam penelitian tersebut hanya terbatas pada 2 variabel saja. Dalam penelitian lain, Mahmudy menggunakan algoritma genetika Terdistribusi dengan Pengkodean Real. Menggunakan uji fungsi dengan 4 variabel, hasilnya algoritma genetika yang dirumuskan menghasilkan hasil yang lebih baik jika dibandingkan dengan algoritma genetika biasa[10]. Dalam menyelesaikan optimasi dengan kendala menggunakan algoritma genetika, Indrianingsih menggunakan algoritma genetika dengan pengkodean bilangan bulat. dalam penelitian tersebut, diterapkan 3 kendala[4]. Selanjutnya, Yongshou dkk. menyelesaikan optimasi fungsi dengan *Adaptive immune-genetic algorithm.* hasilnya, dalam uji beberapa fungsi dengan 3 variabel, modifikasi algoritma genetika yang disingkat AIGA tersebut menyelesaikan optimasi dengan efektif dan efisien[12]. Dalam menyelesaikan optimasi fungsi dengan banyak variabel menggunakan Algoritma Genetika, Mahardhika menggunakan algoritma genetika yang diautomasi. Hasilnya, penelitian tersebut melakukan uji dengan mengoptimasi fungsi mencapai 50 variabel [11]. Ali dan Tawhid menggabungkan algoritma *Particle Swarm Optimizer* dan Algoritma genetika dan melakukan uji optimasi fungsi, mereka mengoptimasi fungsi mencapai 1000 variabel [13]. Selanjutnya, Liu dkk. mengembangkan algoritma genetika yang diberi nama Multi-Agent Genetic Algorithm dan melakukan optimasi fungsi dengan variabel mencapai 10000 variabel [14]. Dalam penelitian-penelitian yang sudah disebutkan diatas, telah dilakukan optimasi fungsi dengan algoritma genetika yang dimodifikasi ataupun digabungkan dengan algoritma lain. namun, belum ada yang mengoptimasi fungsi mencapai lebih dari 10000 variabel. Sedangkan dalam penelitian ini, akan dirumuskan Algoritma

Genetika yang dapat mengoptimasi fungsi dengan minimal 100 variabel dengan target, mampu mengoptimasi fungsi mencapai 100000 variabel dengan akurat.

## 1.2 Rumusan Masalah

Rumusan masalah dalam tugas akhir ini adalah:

1. Bagaimana perumusan algoritma genetika agar dapat menyelesaikan optimasi fungsi dengan banyak variabel.
2. Bagaimana akurasi dari algoritma genetika untuk optimasi fungsi dengan banyak variabel.

## 1.3 Batasan Masalah

Batasan masalah yang digunakan dalam Tugas Akhir ini adalah:

1. Fungsi yang digunakan adalah fungsi tanpa kendala dengan variabel yang saling independen.
2. Jumlah variabel dalam fungsi lebih dari 100.
3. Program dibuat dengan menggunakan bahasa pemrograman Java dalam Apache NetBeans IDE 12.

## 1.4 Tujuan

Tujuan dari pengerjaan Tugas Akhir ini adalah:

1. Merumuskan algoritma genetika agar dapat digunakan untuk menyelesaikan optimasi fungsi dengan banyak variabel.
2. Mengetahui akurasi dari algoritma genetika untuk optimasi fungsi dengan banyak variabel.

## 1.5 Manfaat

Manfaat yang bisa diperoleh dari pengerjaan Tugas Akhir ini adalah Mengetahui seberapa baik algoritma genetika dapat digunakan untuk menyelesaikan Optimasi Fungsi dengan banyak Variabel.

*Halaman ini sengaja dikosongkan*

# BAB II
# TINJAUAN PUSTAKA

## 2.1 Penelitian Terdahulu

Dalam menyelesaikan optimasi fungsi dengan banyak variabel menggunakan Algoritma Genetika, Mahardhika menggunakan algoritma genetika yang diautomasi. Dalam penelitian tersebut, digunakan fungsi Ackley dan fungsi Dejong sebagai fungsi yang dites. Penelitian tersebut melakukan uji dengan mengoptimasi kedua fungsi tersebut mencapai 50 variabel. Hasilnya, algoritma genetika yang diautomasi memberikan hasil yang sangat mendekati solusi eksak yaitu dengan rata-rata error dengan waktu komputasi yang berbeda-beda bergantung pada jenis uji fungsi dan banyaknya variabel[11]. Liu dkk. Selanjutnya, mengembangkan algoritma genetika yang diberi nama Multi-Agent Genetic Algorithm dan melakukan optimasi fungsi dengan variabel mencapai 10000 variabel. Hasilnya, Multi-Agent Genetic Algorithm menghasilkan performa yang baik untuk mengoptimasi fungsi dengan banyak variabel[14].

## 2.2 Algoritma Genetika

Inti dari Algoritma Genetika adalah menyandikan atau mengkodekan suatu optimasi fungsi kedalam sebuah *array* yang berisi karakter untuk merepresentasikan kromosom, kemudian melakukan proses atau operasi manipulasi oleh operator genetika, kemudian dilakukan seleksi berdasarkan nilai *fitness*, dengan target mencari solusi yang optimal berdasarkan masalah yang dicari[3]. Ada dua hal pokok yang harus dipikirkan sebelum menggunakan algoritma genetika,

yang pertama adalah representasi kromosom dan yang kedua adalah pemilihan fungsi $fitness$. Kromosom tersebut akan menjadi solusi untuk optimasi fungsi dan fungsi $fitness$ akan menjadi alat untuk mengukur kebaikan dari suatu kromosom.

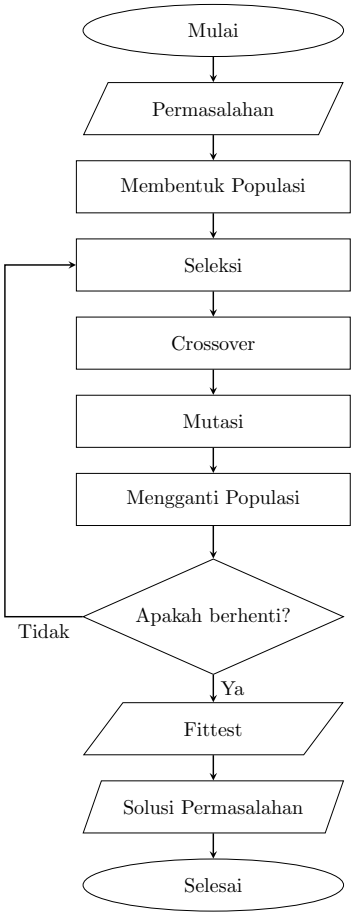
Algoritma genetika dimulai dengan dugaan populasi yang merepresentasikan kromosom-kromosom sebagai solusi permasalahan. Biasanya dugaan tersebut dipilih secara acak yang tersebar di seluruh *search space*. Algoritma genetika kemudian menggunakan tiga operator penting yaitu seleksi, *crossover*, dan mutasi untuk mengolah populasi hingga konvergen atau menemukan hasil yang terbaik [15].

Seleksi berusaha untuk memberikan tekanan pada populasi seperti pada seleksi alami yang ditemukan pada sistem biologis. Kromosom yang buruk akan terbuang dan kromosom yang baik akan memiliki kesempatan yang lebih besar untuk berkembang pada generasi berikutnya. *Crossover* memungkinkan pertukaran informasi pada kromosom untuk membentuk kromosom-kromosom baru. Mutasi digunakan untuk merubah nilai didalam sebuah kromosom secara acak, biasanya dalam satu generasi, mutasi hanya sedikit dilakukan atau kromosom yang dilakukan mutasi hanya sedikit[15].

Setelah seleksi, *crossover*, dan mutasi selesai dilakukan pada populasi awal, maka populasi baru akan terbentuk dan generasi dari populasi baru tersebut telah naik satu tingkat. Proses seleksi, *crossover*, dan mutasi terus dilanjutkan hingga jumlah generasi tertentu telah dilampaui atau kriteria pemberhentian dipenuhi [15]. Langkah-langkah algoritma genetika adalah seperti dalam Gambar 1.

### 2.2.1 Tournament Selection

*Tournament selection* adalah salah satu operator seleksi yang bekerja dengan terlebih dahulu memilih $n$ kromosom dalam populasi secara acak dan kemudian mengambil yang terbaik dari $n$ kromosom tersebut. Dengan




Gambar 2.1: Langkah-langkah Algoritma Genetika

*tournament selection* tersebut, kromosom yang baik akan memiliki kesempatan yang lebih besar untuk terpilih.

### 2.2.2 Average Crossover

*Average Crossover* adalah teknik crossover berdasarkan nilai. *crossover* ini menggunakan dua orang tua untuk menjalankan *crossover* dan hanya menghasilkan satu keturunan. *Average Crossover* menghasilkan keturunan yang didapat dari nilai rata-rata kedua orang tua[16]. Contoh *Average Crossover* dapat dilihat pada Gambar 2.2.

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| orang tua 1 | 5 | 3 | 3 | 2 | 3 | 9 | 7 | 6 | 5 |
| orang tua 2 | 5 | 4 | 7 | 6 | 5 | 2 | 6 | 1 | 3 |
| anak | 5 | 3.5 | 5 | 4 | 4 | 5.5 | 6.5 | 3.5 | 4 |

Gambar 2.2: Contoh *Average Crossover*

### 2.2.3 Desain Awal Kromosom

Dalam penelitian ini, kromosom direpresentasikan dalam bentuk *array*, dengan panjang kromosom adalah sebanyak jumlah variabel atau besar dimensi pada fungsi. Sehingga, desain awal kromosom bisa digambarkan sebagai berikut:

| | | | | |
|---|---|---|---|---|
| $x_1$ | $x_2$ | $x_3$ | ... | $x_n$ |

dengan n sebagai banyaknya variabel.

## 2.3 Algoritma penjumlahan Neumaier

Algoritma penjumlahan Neumaier merupakan pengembangan dari algoritma penjumlahan Kahan atau disebut *compensated algorithm*. Algoritma ini memungkinkan mengurangi nilai error pada operasi penjumlahan dan penguruangan nilai *float* atau *double* pada pemrograman. berikut adalah *pseudocode* dari

algoritma penjumlahan neumaier[17]:

**Algoritma 1:** Neumaier Summation

```
initialize sum = 0, c = 0;
for i ← 1 to input.length do
    t = sum + input[i];
    if |sum| < |input[i]| then
        c += (sum + t) + input[i];
    else
        c += (input[i] - t) + sum;
    end
    sum = t;
end
return sum + c;
```

algoritma ini akan digunakan oleh penulis untuk mengurangi kemungkinan terjadinya error dalam menghitung nilai $fitness$ dalam algoritma genetika.

## 2.4 Fungsi

Berikut ini adalah fungsi yang akan digunakan untuk menguji akurasi dari Algoritma Genetika yang sudah dirumuskan. Dalam penelitian ini, penulis memilih fungsi berdasarkan karakteristik yang berbeda-beda, yaitu berdasarkan besar pangkat, jumlah titik solusi, dan mengandung trigonometri.

1. Fungsi Sphere

$$f(\vec{x}) = \sum_{i=1}^{D} x_i^2 \tag{2.1}$$

dengan $\vec{x} = (x_1, x_2, ..., x_D)$, dimana $-100 \leq x_i \leq 100$, untuk $1 \leq i \leq D$. Minimum global terletak di $x^\star = (0, ..., 0)$ dengan $f(x^\star) = 0$ [1].

2. Fungsi Csendes

$$f(\vec{x}) = \sum_{i=1}^{D} x_i^6 \left(2 + sin(\frac{1}{x_i})\right) \tag{2.2}$$

dengan $\vec{x} = (x_1, x_2, ..., x_D)$, dimana $-1 \leq x_i \leq 1$, untuk $1 \leq i \leq D$. Minimum global terletak di $x^\star = (0, ..., 0)$ dengan $f(x^\star) = 0$ [1].

3. Fungsi Alpine

$$f(\vec{x}) = \sum_{i=1}^{D} |x_i \sin(x_i) + 0.1x_i| \tag{2.3}$$

dengan $\vec{x} = (x_1, x_2, ..., x_D)$, dimana $-10 \leq x_i \leq 10$, untuk $1 \leq i \leq D$. Minimum global terletak di $x^\star = (-9.324, -6.383, -3,041, -0.1, 0, 3.241, 6.183, 9.525)$ dengan $f(x^\star) = 0$ [1].

4. Fungsi *Sum Squares*

$$f(\vec{x}) = \sum_{i=1}^{D} ix_i^2 \tag{2.4}$$

dengan $\vec{x} = (x_1, x_2, ..., x_D)$, dimana $-10 \leq x_i \leq 10$, untuk $1 \leq i \leq D$. Minimum global terletak di $x^\star = (0, ..., 0)$ dengan $f(x^\star) = 0$ [1].

5. Fungsi Schumer Steiglitz

$$f(\vec{x}) = \sum_{i=1}^{D} x_i^4 \tag{2.5}$$

dengan $\vec{x} = (x_1, x_2, ..., x_D)$, dimana $-100 \leq x_i \leq 100$, untuk $1 \leq i \leq D$. Minimum global terletak di $x^\star = (0, ..., 0)$ dengan $f(x^\star) = 0$ [1].

6. Fungsi Quintic

$$f(\vec{x}) = \sum_{i=1}^{D} |x_i^5 - 3x_i^4 + 4x_i^3 + 2x_i^2 - 10x_i - 4| \tag{2.6}$$

dengan $\vec{x} = (x_1, x_2, ..., x_D)$, dimana $-10 \leq x_i \leq 10$, untuk $1 \leq i \leq D$. Minimum global terletak di $x^\star = (-1, -0.4026, 2)$ dengan $f(x^\star) = 0$ [1].

*Halaman ini sengaja dikosongkan*

# BAB III
# METODE PENELITIAN

## 3.1 Studi literatur

Dilakukan studi literatur untuk mendukung pengerjaan Tugas Akhir dan pemahaman yang lebih mendalam mengenai algoritma genetika khususnya untuk optimasi fungsi. Literatur yang dipelajari dapat bersumber dari jurnal, buku, internet, maupun bimbingan dengan dosen pembimbing.

## 3.2 Mencari data

Mencari data berupa fungsi yang digunakan untuk menguji akurasi dari Algoritma Genetika untuk Optimasi Fungsi dengan banyak Variabel. Dalam tugas akhir ini, Algoritma akan diuji dengan sebanyak enam fungsi.

## 3.3 Perumusan Algoritma Genetika untuk optimasi fungsi dengan banyak Variabel

Dalam merumuskan Algoritma Genetika untuk optimasi fungsi dengan banyak Variabel, dirumuskan representasi kromosom, besar populasi, fungsi fitness, operator seleksi, operator crossover, operator mutasi, skema penggantian populasi, dan kondisi pemberhentian.

## 3.4 Perancangan program

Perancangan program dilakukan agar dapat digunakan untuk mengimplementasikan algoritma genetika untuk menyelesaikan optimasi fungsi dengan banyak variabel.

### 3.5 Menghitung Akurasi

Program yang sudah dirancang akan diuji menggunakan fungsi yang sudah didapat dan akan dihitung akurasi dari hasil program.

### 3.6 Penarikan Kesimpulan

Penarikan kesimpulan dilakukan dengan memperhatikan hasil dan pembahasan yang telah diselesaikan pada tahap-tahap sebelumnya. Kesimpulan yang ditarik adalah mengenai apakah algoritma genetika dapat digunakan untuk menyelesaikan optimasi fungsi dengan banyak variabel.

# BAB IV
# ANALISIS DAN PEMBAHASAN

## 4.1 Perumusan Algoritma Genetika untuk Optimasi Fungsi dengan Banyak Variabel

Algoritma Genetika untuk Optimasi Fungsi dengan Banyak Variabel memiliki karakteristik sebagai berikut:

1. Representasi kromosom dari Algoritma ini adalah solusi optimum sebanyak jumlah variabel dari fungsi yang diuji, dengan nilai awal pada kromosom adalah bilangan acak dengan rentang sesuai dengan tetapan dalam fungsi yang diuji. Sebagai contoh, dalam fungsi *Sphere* pada persamaan 2.1, maka nilai awal pada kromosom adalah nilai acak antara -100 sampai 100 sebanyak jumlah variabel.

2. Besar populasi yang digunakan adalah sebanyak 25, 50, 100, 200 dan 400. kelima besaran populasi tersebut dipilih untuk menentukan apakah semakin besar populasi mempengaruhi hasil dari algoritma yang telah dirumuskan.

3. Fungsi *fitness* yang digunakan adalah fungsi yang akan diuji. Dalam algoritma genetika, setelah melalui proses *crossover* dan mutasi untuk mendapatkan keturunan, dilakukan perhitungan nilai *fitness* dari keturunan tersebut. Namun dalam penelitian ini, perhitungan nilai *fitness* dilakukan ketika proses *crossover* dan mutasi berjalan, nilai *fitness* dihitung beriringan dengan didapatnya nilai baru pada suatu keturunan. Nilai

*fitness* tersebut didapat dengan cara perhitungan nilai *fitness* pada nilai baru dikurangi perhitungan nilai *fitness* pada nilai lama. Sebagai contoh dalam fungsi *sphere* pada persamaan 2.1, misal nilai $a$ adalah nilai baru setelah proses *crossover* atau mutasi dan nilai $b$ adalah nilai sebelum proses *crossover* atau mutasi, maka fungsi *fitness* dalam satu nilai baru didapat dengan cara:

$$fitness(baru) = fitness(lama) + a^2 - b^2$$

sehingga, setelah proses *crossover* dan mutasi, tidak perlu dilakukan perhitungan nilai *fitness* karena keturunan sudah memiliki nilai *fitness* baru. Dengan cara perhitungan seperti ini, didapat hasil $fitness$ yang sama jika dibandingkan dengan cara perhitungan ulang tiap setelah operasi dilakukan.

4. Seleksi yang digunakan adalah *Tournament Selection*, seleksi tersebut dipilih karena memungkinkan terpilihnya kromosom yang baik untuk proses selanjutnya.

5. *Crossover* yang digunakan adalah *Average Crossover*, *crossover* tersebut dipilih karena operator *crossover* tersebut memiliki teknik yang tepat sesuai dengan representasi kromosom. Dalam proses perhitungan *Average Crossover*, keturunan atau dalam penelitian ini disebut anak akan membawa informasi kromosom dan nilai $fitness$ milik orangtua pertama. Kemudian dilakukan *crossover* dengan menghitung rata-rata dari kromosom anak dan orangtua kedua. Perhitungan tersebut dilakukan berurutan sesuai dengan urutan pada isi kromosom. Setelah perhitungan dan didapat nilai baru pada kromosom, dilakukan pembandingan nilai fitness baru dengan nilai $fitness$ sebelum proses

*crossover*. Bila nilai $fitness$ baru lebih rendah dari nilai $fitness$ sebelum perhitungan, maka nilai baru pada kromosom dipertahankan. Sebaliknya, bila nilai $fitness$ baru lebih tinggi dibanding nilai $fitness$ sebelum proses perhitungan, maka nilai baru pada kromosom tidak dipertahankan dan dikembalikan seperti semula. Sebagai contoh, akan dilakukan proses *crossover* dengan fungsi $fitness$ adalah fungsi 2.1 pada 2.4. Proses perhitungan *Average Crossover* dengan kromosom sepanjang tiga variabel diilustrasikan sebagai berikut:
Kromosom:

| | | | |
|---|---|---|---|
| Orang tua 1 | 2 | 8 | 5 |
| Orang tua 2 | 4 | 6 | 1 |
| Anak | 2 | 8 | 5 |

Dengan nilai *fitness* Orangtua 1 adalah 93 dan nilai *fitness* Orangtua 2 adalah 53.

(a) Proses pada urutan pertama:
Pada urutan pertama, masing-masing nilai pada kromosom Anak dan Orangtua 2 adalah

| | |
|---|---|
| Orangtua 2 | 2 |
| Anak | 4 |

Sehingga, setelah proses *crossover*, nilai anak menjadi

| | | | |
|---|---|---|---|
| Kromosom : | 3 | 8 | 5 |

Selanjutnya dihitung nilai *fitness* dari anak, yaitu

$$\begin{aligned} fitness(baru) &= 93 + 9 - 4 \\ &= 98 \end{aligned}$$

karena nilai *fitness* baru pada Anak lebih tinggi dari nilai *fitness* sebelum proses crossover, maka nilai pada urutan pertama tidak berubah. Sehingga setelah proses *crossover* pada urutan pertama, kromosom Anak adalah

Kromosom :

| | | |
|---|---|---|
| 2 | 8 | 5 |

(b) Proses pada urutan kedua:
Pada urutan kedua, masing-masing nilai pada kromosom Anak dan Orangtua 2 adalah

| | |
|---|---|
| Orangtua 2 | 6 |
| Anak | 8 |

Sehingga, setelah proses *crossover*, nilai anak menjadi

Kromosom :

| | | |
|---|---|---|
| 2 | 7 | 5 |

Selanjutnya dihitung nilai *fitness* dari anak, yaitu

$$\begin{aligned} fitness(baru) &= 93 + 49 - 64 \\ &= 78 \end{aligned}$$

karena nilai *fitness* baru pada Anak lebih rendah dari nilai *fitness* sebelum proses *crossover*, maka nilai baru pada kromosom berubah. Sehingga kromosom Anak setelah proses *crossover* pada

Kromosom :

| | | |
|---|---|---|
| 2 | 7 | 5 |

urutan kedua adalah

(c) Proses pada urutan ketiga:
Pada urutan ketiga, masing-masing nilai pada kromosom Anak dan Orangtua 2 adalah

| | |
|---|---|
| Orangtua 2 | 1 |
| Anak | 5 |

Sehingga, setelah proses *crossover*, nilai anak menjadi

Kromosom :

| | | |
|---|---|---|
| 2 | 7 | 3 |

Selanjutnya dihitung nilai *fitness* dari anak, yaitu

$$\begin{aligned} fitness(baru) &= 78 + 9 - 25 \\ &= 62 \end{aligned}$$

karena nilai *fitness* baru pada Anak lebih rendah dari nilai *fitness* sebelum proses *crossover*, maka nilai baru pada kromosom berubah. Sehingga kromosom Anak setelah proses *crossover* pada urutan ketiga adalah

Kromosom :

| | | |
|---|---|---|
| 2 | 7 | 3 |

Sehingga didapat kromosom baru setelah proses *crossover* yaitu

Kromosom :

| | | |
|---|---|---|
| 2 | 7 | 3 |

dengan nilai *fitness* dari kromosom tersebut adalah 62.

6. Mutasi dilakukan dengan cara mengubah suatu nilai dalam kromosom menjadi suatu bilangan acak lain dengan cara nilai pada kromosom ditambahkan nilai acak antara -0,5 sampai 0,5. Proses terjadinya mutasi pada kromosom terjadi apabila program membangkitkan nilai acak dibawah peluang mutasi yang ditentukan. Dalam penelitian ini, ditentukan peluang mutasi sebanyak 0,05 dan 1. Artinya ketika ditentukan peluang mutasi sebesar 0,05, mutasi pada suatu kromosom akan terjadi apabila program membangkitkan nilai acak dibawah 0,05, ketika peluang mutasi ditetapkan sebesar 1, artinya mutasi pada suatu kromosom akan terjadi pada seluruh kromosom dalam populasi. Kedua besar peluang mutasi tersebut dipilih karena biasanya dalam satu generasi, hanya sedikit kromosom yang dilakukan mutasi[15], Dalam hal ini, agar hanya sedikit jumlah kromosom yang dilakukan mutasi, maka dalam rentang peluang 0 sampai 1, dipilih nilai dibawah 0,1 sehingga dipilih nilai 0,05. Maka, dalam penelitian ini, akan dibandindingkan hasil apabila mutasi dilakukan hanya pada sedikit kromosom, dengan mutasi dilakukan pada seluruh kromosom. Proses mutasi pada suatu kromsom diilustrasikan sebagai berikut:

Kromosom :

| | | |
|---|---|---|
| 3 | 9 | 2 |

dengan nilai *fitness* adalah 104, satu per satu isi kromosom kemudian akan ditambahkan dengan nilai acak antara -0,5 sampai 0,5.

(a) Proses pada urutan pertama:

misal nilai acak adalah 0,3. Maka kromosom menjadi:

Kromosom :

| | | |
|---|---|---|
| 3,3 | 9 | 2 |

Selanjutnya dihitung nilai *fitness* dari kromosom, yaitu

$$\begin{aligned} fitness(baru) &= 104 + 10,89 - 9 \\ &= 105,89 \end{aligned}$$

karena nilai *fitness* baru lebih besar dari nilai *fitness* sebelum proses mutasi, maka nilai pertama pada kromosom tidak diubah, sehingga kromosom tetap seperti sebelumnya.

(b) Proses pada urutan kedua:
misal nilai acak adalah -0,2. Maka kromosom menjadi:

Kromosom :

| | | |
|---|---|---|
| 3 | 8,8 | 2 |

Selanjutnya dihitung nilai *fitness* dari kromosom, yaitu

$$\begin{aligned} fitness(baru) &= 104 + 77,4 - 81 \\ &= 100,44 \end{aligned}$$

karena nilai *fitness* baru lebih kecil dari nilai *fitness* sebelum proses mutasi, maka nilai kedua pada kromosom diubah menjadi nilai yang baru, sehingga kromosom menjadi:

Kromosom :

| | | |
|---|---|---|
| 3 | 8,8 | 2 |

(c) Proses pada urutan ketiga:
misal nilai acak adalah 0,1. Maka kromosom menjadi:

| Kromosom : | 3 | 8,8 | 2,1 |
|---|---|---|---|

Selanjutnya dihitung nilai *fitness* dari kromosom, yaitu

$$\begin{aligned} fitness(baru) &= 100,44 + 4,41 - 4 \\ &= 100,82 \end{aligned}$$

karena nilai *fitness* baru lebih besar dari nilai *fitness* sebelum proses mutasi, maka nilai ketiga pada kromosom tidak diubah, sehingga kromosom tetap seperti sebelumnya.

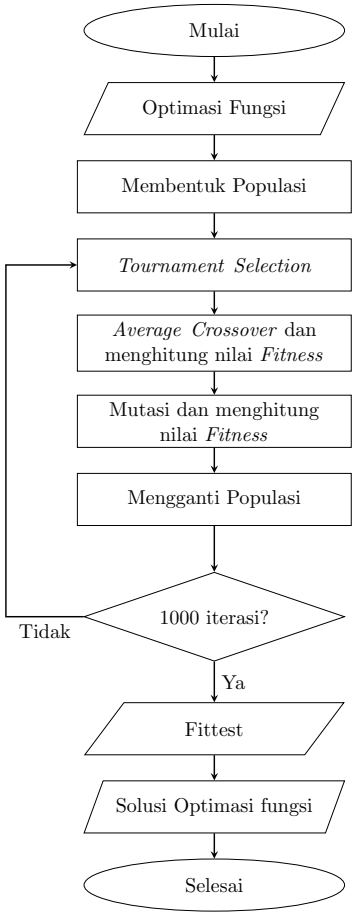
maka, kromosom setelah proses mutasi adalah

| Kromosom : | 3 | 8,8 | 2 |
|---|---|---|---|

dengan nilai *fitness* adalah 100,44.

7. Kondisi pemberhentian dalam Algoritma ini adalah setelah mencapai 1000 generasi

Berdasarkan perumusan algoritma tersebut, langkah-langkah Algoritma Genetika untuk Optimasi Fungsi dengan Banyak Variabel ditunjukkan pada Gambar 4.1




Gambar 4.1: Langkah-langkah Algoritma Genetika untuk Optimasi Fungsi dengan Banyak Variabel

## 4.2 Implementasi Pada Program

Agar dapat mengetahui akurasi dari Algoritma Genetika untuk Optimasi Fungsi dengan Banyak Variabel, dilakukan perancangan program yang mengimplementasikan Algoritma

Genetika yang telah dirumuskan sebelumnya.

Untuk mengimplementasikan Algoritma Genetika untuk Optimasi Fungsi dengan banyak Variabel, digunakan beberapa *class* sebagai berikut:

1. Library.java
   *Class* ini adalah *class* yang menyimpan jumlah atau banyaknya variabel dari fungsi yang akan dioptimalkan. Jumlah atau banyaknya variabel ini akan menjadi banyaknya isi suatu kromosom. Keseluruhan *source code* dapat dilihat pada Lampiran A

2. Solution.java
   *Class* ini adalah *class* yang digunakan untuk merepresentasikan kromosom dalam algoritma genetika. Dalam *class* ini terdapat fungsi yang digunakan untuk mengolah kromosom dan *fitness* dari kromosom. Suatu kromosom dibentuk dari bilangan-bilangan acak dengan menjalankan *source code* sebagai berikut:

   ```
   public Solution(){
       for(int i = 0; i < x.length; i++){
           x[i] = Math.random();
   }
   ```

   rentang bilangan acak bergantung pada tetapan dalam fungsi pada 2.4. Selain itu, kelas ini juga memiliki *function* yang digunakan untuk memperoleh informasi yang berisi nilai terbaik dalam kromosom, dan rata-rata error dari solusi yang dihasilkan. Keseluruhan *source code* dapat dilihat pada Lampiran 2

3. Population.java
   *Class* ini adalah *class* yang digunakan untuk mendefinisikan populasi dalam algoritma genetika Dalam *class* ini terdapat fungsi yang digunakan

untuk mengolah populasi dan mendapatkan kromosom terbaik dalam populasi. Suatu populasi dibentuk dari kromosom-kromosom dengan menjalankan *source code* sebagai berikut:

```
public Population(int numberOfSolutions,
    boolean initialization){
     solution = new Solution[
        numberOfSolutions];
     if(initialization){
         for(int i = 0; i<numberOfSolutions;
            i++){
             Solution newSolution = new
                Solution();
             saveSolution(i,newSolution);
         }
     }
}
```

untuk mendapatkan kromosom terbaik dalam populasi, digunakan *source code* sebagai berikut:

```
public Solution getFittest(){
    Solution fittest = solution[0];
    for(int i = 0; i<solution.length; i++){
        if(getSolution(i).getFitness()<
            fittest.getFitness()){
             fittest = getSolution(i);
        }
    }
    return fittest;
}
```

Keseluruhan *source code* dapat dilihat pada Lampiran 3

4. GeneticAlgorithm.java
   *Class* ini adalah *class* yang digunakan untuk mengimplementasikan algoritma genetika. Dalam *class*

ini terdapat beberapa fungsi yang digunakan untuk melakukan *average crossover*, mutasi, *tournament selection*. *average crossover* dilakukan seperti *source code* berikut:

```
public static Solution crossover(Solution
    parent1, Solution parent2){
     Solution child1 = new Solution(parent1.
         getX(), parent1.getFitness());
     for(int i = 0; i < Library.
         getNumOfVariables(); i++){
          double f = child1.getFitness();
          child1.setX(i, (parent1.getX(i) +
              parent2.getX(i))/2);
          if(child1.getFitness() > f){
               child1.setX(i, parent1.getX(i));
          }
     }
     return child1;
}
```

mutasi dilakukan dalam *source code* berikut:

```
public static void mutation(Solution
    solution){
     Solution solution2 = new Solution(
         solution.getX(), solution.getFitness
         ());
     for(int i = 0; i < Library.
         getNumOfVariables(); i++){
          double f = solution.getFitness();
          solution.setX(i, solution.getX(i)+
              Math.random()-0.5);
          if(solution.getFitness() > f){
               solution.setX(i, solution2.getX(
                   i));
          }
     }
}
```

*tournament selection* dilakukan dalam *source code* berikut:

```
public static Solution tournament(Population
    pop){
    Population tournamentselection = new
       Population(tournamentsize, false);
    for(int i = 0; i < tournamentsize; i++){
        int random = (int)(Math.random()*pop
            .popSize());
        tournamentselection.saveSolution(i,
           pop.getSolution(random));
    }
    Solution fittest = tournamentselection.
       getFittest();
    return fittest;
}
```

Keseluruhan *source code* dapat dilihat pada Lampiran 4

5. Main.java
   *Class* ini adalah *class* utama yang digunakan untuk menjalankan program. Keseluruhan *source code* Main.java dapat dilihat pada lampiran 5.

## 4.3 Akurasi hasil Perumusan Algoritma Genetika untuk Optimasi Fungsi dengan Banyak Variabel

Program yang sudah disusun kemudian diuji oleh enam fungsi yang sudah ditentukan dalam 2.4. Dalam pengujian tersebut akan ditentukan akurasi dari algoritma genetika yang telah dirumuskan. Akurasi ditinjau dari keluaran atau hasil dari jalannya program yang merupakan nilai $fitness$ dari kromosom dalam algoritma genetika, dan nilai $fitness$ tersebut juga merupakan global optimum dari fungsi yang diuji. Dalam penelitian ini, penulis menentukan parameter baiknya hasil apabila error dari nilai $fitness$ berdasarkan nilai global optimum tidak lebih dari 0,001 atau $1 \times 10^{-3}$.

Dalam pengujian pada program dengan fungsi-fungsi yang telah ditentukan, ditentukan hasil pada program berdasarkan jumlah variabel, besar populasi, dan peluang mutasi yang berbeda. Jumlah variabel yang diuji sebanyak 100, 1000, 10000, dan 100000 variabel, besar populasi sebesar 25, 50, 100, 200, dan 400, dan peluang mutasi sebesar 0,05 dan 1. Program yang telah disusun, dijalankan dalam perangkat komputer dengan spesifikasi *processor* intel *core* i3-6100U dengan kapasitas memori sebesar 4GB.

### 4.3.1 Uji fungsi Sphere

Tabel 4.1: Hasil uji pada fungsi sphere

| Jumlah Variabel | Besar Populasi | Peluang Mutasi | Nilai Fitness | Waktu (detik) |
|---|---|---|---|---|
| 100 | 25 | 0,05 | $1.315\times10^{-6}$ | 0,205 |
| | | 1 | $8,654\times10^{-10}$ | 0,328 |
| | 50 | 0,05 | $4,096\times10^{-8}$ | 0,271 |
| | | 1 | $-2,516\times10^{-10}$ | 0,619 |
| | 100 | 0,05 | $4,018\times10^{-9}$ | 0,489 |
| | | 1 | $-3,528\times10^{-10}$ | 1,079 |
| | 200 | 0,05 | $-3,944\times10^{-10}$ | 0,87 |
| | | 1 | $-4,341\times10^{-10}$ | 2,009 |
| | 400 | 0,05 | $-4,675\times10^{-10}$ | 1,641 |
| | | 1 | $-3,985\times10^{-10}$ | 3,691 |
| 1000 | 25 | 0,05 | $3,736\times10^{-5}$ | 0,858 |
| | | 1 | $2,386\times10^{-8}$ | 2,238 |
| | 50 | 0,05 | $2,103\times10^{-6}$ | 1,612 |
| | | 1 | $-1,097\times10^{-8}$ | 4,4 |
| | 100 | 0,05 | $8,399\times10^{-8}$ | 3,073 |
| | | 1 | $-1,168\times10^{-8}$ | 8,686 |

Tabel 4.1 Hasil uji pada fungsi sphere (lanjutan)

| Jumlah Variabel | Besar Populasi | Peluang Mutasi | Nilai Fitness | Waktu (detik) |
|---|---|---|---|---|
| 10000 | 200 | 0,05 | $-9{,}757\times10^{-9}$ | 6,503 |
| | | 1 | $-1{,}030\times10^{-8}$ | 18,155 |
| | 400 | 0,05 | $-1{,}409\times10^{-8}$ | 12,017 |
| | | 1 | $-1{,}524\times10^{-8}$ | 33,28 |
| | 25 | 0,05 | $2{,}683\times10^{-4}$ | 7,368 |
| | | 1 | $1{,}066\times10^{-7}$ | 21,376 |
| | 50 | 0,05 | $2{,}736\times10^{-5}$ | 14,221 |
| | | 1 | $-1{,}512\times10^{-7}$ | 42,894 |
| | 100 | 0,05 | $4{,}015\times10^{-7}$ | 29,413 |
| | | 1 | $-2{,}888\times10^{-7}$ | 85,528 |
| | 200 | 0,05 | $-3{,}458\times10^{-7}$ | 70,12 |
| | | 1 | $-2{,}809\times10^{-7}$ | 181,304 |
| | 400 | 0,05 | $-3{,}648\times10^{-7}$ | 140,551 |
| | | 1 | $-4{,}501\times10^{-7}$ | 352,238 |
| 100000 | 25 | 0,05 | 0,002 | 77,892 |
| | | 1 | $-7{,}331\times10^{-6}$ | 243,157 |
| | 50 | 0,05 | $2{,}366\times10^{-4}$ | 140,817 |
| | | 1 | $-9{,}235\times10^{-6}$ | 495,729 |
| | 100 | 0,05 | $2{,}607\times10^{-6}$ | 384,877 |
| | | 1 | $-1{,}002\times10^{-5}$ | 1079,571 |
| | 200 | 0,05 | $-1{,}381\times10^{-5}$ | 704,636 |
| | | 1 | $-1{,}555\times10^{-5}$ | 2010,074 |
| | 400 | 0,05 | $-1{,}118\times10^{-5}$ | 1534,31 |
| | | 1 | $-1{,}247\times10^{-5}$ | 4241,712 |

Berdasarkan Tabel 4.1, program yang telah disusun mendapatkan hasil yang baik dalam menyelesaikan masalah optimasi fungsi dengan banyak variabel pada uji fungsi *sphere*, hal ini dikarenakan nilai $fitness$ yang dihasilkan oleh

program memiliki *error* lebih kecil dari 0,001. Hanya didapat hasil yang kurang baik pada uji 100000 variabel dengan besar populasi 25 dan peluang mutasi sebesar 0,05.

### 4.3.2 Uji fungsi Csendes

Tabel 4.2: Hasil uji pada fungsi csendes

| Jumlah Variabel | Besar Populasi | Peluang Mutasi | Nilai Fitness | Waktu (detik) |
|---|---|---|---|---|
| 100 | 25 | 0,05 | $-1{,}936\times10^{-14}$ | 0,632 |
| | | 1 | $-1{,}879\times10^{-14}$ | 1,279 |
| | 50 | 0,05 | $-2{,}645\times10^{-14}$ | 1,182 |
| | | 1 | $-2{,}631\times10^{-14}$ | 2,268 |
| | 100 | 0,05 | $-1{,}946\times10^{-14}$ | 1,945 |
| | | 1 | $-2{,}624\times10^{-20}$ | 4,339 |
| | 200 | 0,05 | $-2{,}586\times10^{-14}$ | 3,626 |
| | | 1 | $-3{,}268\times10^{-14}$ | 8,322 |
| | 400 | 0,05 | $-3{,}067\times10^{-14}$ | 6,993 |
| | | 1 | $-2{,}843\times10^{-14}$ | 16,342 |
| 1000 | 25 | 0,05 | $-3{,}666\times10^{-13}$ | 3,249 |
| | | 1 | $-4{,}510\times10^{-13}$ | 7,991 |
| | 50 | 0,05 | $-8{,}682\times10^{-13}$ | 6,154 |
| | | 1 | $-7{,}190\times10^{-13}$ | 15,898 |
| | 100 | 0,05 | $-8{,}937\times10^{-13}$ | 12,495 |
| | | 1 | $-6{,}493\times10^{-13}$ | 34,509 |
| | 200 | 0,05 | $-9{,}835\times10^{-13}$ | 29,136 |
| | | 1 | $-9.283\times10^{-13}$ | 71,198 |
| | 400 | 0,05 | $-9{,}514\times10^{-13}$ | 59,094 |
| | | 1 | $-9{,}625\times10^{-13}$ | 145,931 |
| 10000 | 25 | 0,05 | $-2{,}705\times10^{-11}$ | 30,285 |
| | | 1 | $-2{,}581\times10^{-11}$ | 67,745 |

Tabel 4.2 Hasil uji pada fungsi csendes (lanjutan)

| Jumlah Variabel | Besar Populasi | Peluang Mutasi | Nilai Fitness | Waktu (detik) |
|---|---|---|---|---|
| 100000 | 50 | 0,05 | -2,931$\times 10^{-11}$ | 49,084 |
| | | 1 | -2,722$\times 10^{-11}$ | 141,063 |
| | 100 | 0,05 | -2,086$\times 10^{-11}$ | 101,804 |
| | | 1 | -2,646$\times 10^{-11}$ | 304,769 |
| | 200 | 0,05 | -2,928$\times 10^{-11}$ | 232,681 |
| | | 1 | -2,530$\times 10^{-7}$ | 617,629 |
| | 400 | 0,05 | -2,751$\times 10^{-11}$ | 482,055 |
| | | 1 | -2.668$\times 10^{-11}$ | 1271,316 |
| | 25 | 0,05 | -9,110$\times 10^{-10}$ | 296,004 |
| | | 1 | -5,629$\times 10^{-10}$ | 644.571 |
| | 50 | 0,05 | -1,050$\times 10^{-9}$ | 466,459 |
| | | 1 | -8,942$\times 10^{-10}$ | 1311,079 |
| | 100 | 0,05 | -1,104$\times 10^{-9}$ | 1050,364 |
| | | 1 | -1,032$\times 10^{-9}$ | 3014,559 |
| | 200 | 0,05 | -8,079$\times 10^{-10}$ | 2293,353 |
| | | 1 | -9,545$\times 10^{-10}$ | 6585,071 |
| | 400 | 0,05 | -1,000$\times 10^{-9}$ | 5285,716 |
| | | 1 | -1,502$\times 10^{-5}$ | 14047,582 |

Berdasarkan Tabel 4.2, program yang telah disusun mendapatkan hasil yang baik dalam menyelesaikan masalah optimasi fungsi dengan banyak variabel pada uji fungsi *csendes*, hal ini dikarenakan nilai $fitness$ yang dihasilkan oleh program memiliki *error* lebih kecil dari 0,001.

### 4.3.3 Uji fungsi Alpine

Tabel 4.3: Hasil uji pada fungsi alpine

| Jumlah Variabel | Besar Populasi | Peluang Mutasi | Nilai Fitness | Waktu (detik) |
|---|---|---|---|---|
| 100 | 25 | 0,05 | 0,027 | 0,498 |
| | | 1 | 0,002 | 0,739 |
| | 50 | 0,05 | 0,003 | 0,677 |
| | | 1 | $5{,}998\times10^{-4}$ | 1,435 |
| | 100 | 0,05 | $4{,}125\times10^{-4}$ | 1,357 |
| | | 1 | $1{,}988\times10^{-4}$ | 2,782 |
| | 200 | 0,05 | $2{,}112\times10^{-5}$ | 2,527 |
| | | 1 | $1{,}725\times10^{-5}$ | 5,475 |
| | 400 | 0,05 | $6{,}429\times10^{-8}$ | 4,952 |
| | | 1 | $5{,}156\times10^{-8}$ | 10,827 |
| 1000 | 25 | 0,05 | 1,120 | 2,843 |
| | | 1 | 0,121 | 6,424 |
| | 50 | 0,05 | 0,632 | 5,621 |
| | | 1 | 0,053 | 13,055 |
| | 100 | 0,05 | 0,295 | 11,247 |
| | | 1 | 0,022 | 26,29 |
| | 200 | 0,05 | 0,088 | 23,227 |
| | | 1 | 0,004 | 52,941 |
| | 400 | 0,05 | 0,018 | 46,259 |
| | | 1 | 0,001 | 105,716 |
| 10000 | 25 | 0,05 | 14,108 | 27,366 |
| | | 1 | 1,578 | 62,75 |
| | 50 | 0,05 | 9,080 | 55,287 |
| | | 1 | 0,797 | 128,134 |
| | 100 | 0,05 | 5,135 | 113,174 |
| | | 1 | 0,301 | 266,233 |

Tabel 4.3 Hasil uji pada fungsi alpine (lanjutan)

| Jumlah Variabel | Besar Populasi | Peluang Mutasi | Nilai Fitness | Waktu (detik) |
|---|---|---|---|---|
| 100000 | 200 | 0,05 | 2,156 | 239,403 |
| | | 1 | 0,096 | 536,89 |
| | 400 | 0,05 | 0,923 | 468,177 |
| | | 1 | 0,037 | 1062,39 |
| | 25 | 0,05 | 143,956 | 274,347 |
| | | 1 | 16,718 | 618,282 |
| | 50 | 0,05 | 87,733 | 546,849 |
| | | 1 | 8,269 | 1257,861 |
| | 100 | 0,05 | 52,406 | 1205,146 |
| | | 1 | 3,485 | 2724,493 |
| | 200 | 0,05 | 25,803 | 2582,714 |
| | | 1 | 1,192 | 6022,949 |
| | 400 | 0,05 | 11,241 | 5719,428 |
| | | 1 | 0,466 | 12779,271 |

Berdasarkan Tabel 4.3, uji pada fungsi *alpine* hanya mendapatkan hasil yang baik pada uji 100 variabel dengan besar populasi 100, 200, dan 400. Hal tersebut dikarenakan nilai *fitness* yang dihasilkan memiliki *error* kurang dari 0,001. Ketika kondisi besar populasi 50, hanya didapat hasil yang baik ketika peluang mutasi sebesar 1.

### 4.3.4 Uji fungsi Sum Squares

Tabel 4.4: Hasil uji pada fungsi sum squares

| Jumlah Variabel | Besar Populasi | Peluang Mutasi | Nilai Fitness | Waktu (detik) |
|---|---|---|---|---|
| | 25 | 0,05 | $1{,}073\times10^{-5}$ | 0,185 |
| | | 1 | $1{,}419\times10^{-8}$ | 0,331 |

Tabel 4.4 Hasil uji pada fungsi sum squares (lanjutan)

| Jumlah Variabel | Besar Populasi | Peluang Mutasi | Nilai Fitness | Waktu (detik) |
|---|---|---|---|---|
| 1000 | 50 | 0,05 | $1{,}197\times10^{-6}$ | 0,257 |
| | | 1 | $1{,}707\times10^{-9}$ | 0,595 |
| | 100 | 0,05 | $6{,}256\times10^{-9}$ | 0,475 |
| | | 1 | $-1{,}962\times10^{-10}$ | 1,072 |
| | 200 | 0,05 | $-2{,}762\times10^{-10}$ | 0,91 |
| | | 1 | $-1{,}792\times10^{-10}$ | 1,957 |
| | 400 | 0,05 | $-2.003\times10^{-10}$ | 1,49 |
| | | 1 | $-2.112\times10^{-10}$ | 3,812 |
| 10000 | 25 | 0,05 | 0,007 | 0,835 |
| | | 1 | $1{,}145\times10^{-5}$ | 2,224 |
| | 50 | 0,05 | $4{,}758\times10^{-4}$ | 1,565 |
| | | 1 | $3{,}879\times10^{-7}$ | 4,507 |
| | 100 | 0,05 | $3{,}850\times10^{-6}$ | 3,123 |
| | | 1 | $3{,}303\times10^{-8}$ | 8,797 |
| | 200 | 0,05 | $-3{,}952\times10^{-8}$ | 6,471 |
| | | 1 | $-9{,}290\times10^{-8}$ | 17,703 |
| | 400 | 0,05 | $-5{,}526\times10^{-8}$ | 12,03 |
| | | 1 | $-6{,}305\times10^{-8}$ | 34,516 |
| | 25 | 0,05 | 1,055 | 7,239 |
| | | 1 | 0,001 | 21,188 |
| | 50 | 0,05 | 0,047 | 13,766 |
| | | 1 | $2{,}164\times10^{-5}$ | 43,666 |
| | 100 | 0,05 | $5{,}497\times10^{-4}$ | 28,923 |
| | | 1 | $-1{,}752\times10^{-5}$ | 88,827 |
| | 200 | 0,05 | $-1{,}540\times10^{-5}$ | 65,493 |
| | | 1 | $-1{,}733\times10^{-5}$ | 187,82 |
| | 400 | 0,05 | $-2{,}111\times10^{-5}$ | 139,729 |
| | | 1 | $-1{,}855\times10^{-5}$ | 360,447 |

Tabel 4.4 Hasil uji pada fungsi sum squares (lanjutan)

| Jumlah Variabel | Besar Populasi | Peluang Mutasi | Nilai Fitness | Waktu (detik) |
|---|---|---|---|---|
| 100000 | 25 | 0,05 | 97,683 | 71,488 |
| | | 1 | 0,153 | 231,151 |
| | 50 | 0,05 | 4,451 | 132,052 |
| | | 1 | $-1{,}227\times10^{-5}$ | 425,643 |
| | 100 | 0,05 | 0,076 | 321,024 |
| | | 1 | -0,005 | 1025,982 |
| | 200 | 0,05 | -0,005 | 709,078 |
| | | 1 | -0,006 | 2040,865 |
| | 400 | 0,05 | -0.006 | 1434.41 |
| | | 1 | -0.006 | 4220.962 |

Berdasarkan Tabel 4.4, algoritma yang telah disusun mendapatkan hasil yang baik dalam seluruh uji variabel. Namun, hanya dalam kondisi besar populasi dan besar peluang mutasi tertentu. pada uji 100 variabel, didapat hasil yang baik dalam semua kondisi, pada uji 1000 variabel, didapat hasil yang kurang baik atau nilai *fitness* yang dihasilkan memiliki *error* diatas 0,001 hanya ketika kondisi besar populasi 25 dan besar peluang mutasi 0,05. Pada uji 10000 variabel, didapat hasil yang baik pada kondisi besar populasi 100, 200, dan 400. Ketika besar populasi 50, didapat hasil yang baik ketika besar peluang mutasi sebesar 1. Pada uji 100000 variabel, hanya didapat hasil yang baik atau nilai *fitness* yang dihasilkan memiliki *error* dibawah 0,001 ketika besar populasi sebesar 50 dan peluang mutasi sebesar 1.

### 4.3.5 Uji fungsi Schumer-steiglitz

Tabel 4.5: Hasil uji pada fungsi schumer-steiglitz

| Jumlah Variabel | Besar Populasi | Peluang Mutasi | Nilai Fitness | Waktu (detik) |
|---|---|---|---|---|
| 100 | 25 | 0,05 | $-1{,}176\times10^{-6}$ | 0,179 |
| | | 1 | $-1{,}828\times10^{-6}$ | 0,373 |
| | 50 | 0,05 | $-1{,}543\times10^{-6}$ | 0,285 |
| | | 1 | $-1{,}678\times10^{-6}$ | 0.6 |
| | 100 | 0,05 | $-2{,}720\times10^{-6}$ | 0,466 |
| | | 1 | $-1{,}727\times10^{-6}$ | 1,04 |
| | 200 | 0,05 | $-2{,}372\times10^{-6}$ | 0,744 |
| | | 1 | $-2{,}104\times10^{-6}$ | 1,897 |
| | 400 | 0,05 | $-2{,}274\times10^{-6}$ | 1,263 |
| | | 1 | $-2{,}485\times10^{-6}$ | 3,758 |
| 1000 | 25 | 0,05 | $-4{,}255\times10^{-5}$ | 0,919 |
| | | 1 | $-5{,}443\times10^{-5}$ | 2,23 |
| | 50 | 0,05 | $-5{,}244\times10^{-5}$ | 1,497 |
| | | 1 | $-7{,}777\times10^{-5}$ | 4,262 |
| | 100 | 0,05 | $-5{,}339\times10^{-5}$ | 2,613 |
| | | 1 | $-5{,}470\times10^{-5}$ | 8,52 |
| | 200 | 0,05 | $-5{,}850\times10^{-5}$ | 4,895 |
| | | 1 | $-5{,}915\times10^{-5}$ | 17,146 |
| | 400 | 0,05 | $-6{,}089\times10^{-5}$ | 9,501 |
| | | 1 | $-7{,}174\times10^{-5}$ | 34,187 |
| 10000 | 25 | 0,05 | -0,001 | 7,323 |
| | | 1 | -0,001 | 22,649 |
| | 50 | 0,05 | -0,001 | 12,864 |
| | | 1 | -0,002 | 43,792 |
| | 100 | 0,05 | -0,003 | 24,401 |
| | | 1 | -0,001 | 89,69 |

Tabel 4.5 Hasil uji pada fungsi schumer-steiglitz (lanjutan)

| Jumlah Variabel | Besar Populasi | Peluang Mutasi | Nilai Fitness | Waktu (detik) |
|---|---|---|---|---|
| 100000 | 200 | 0,05 | -0,002 | 48,5 |
| | | 1 | -0,002 | 191,3 |
| | 400 | 0,05 | -0,002 | 119,825 |
| | | 1 | -0,003 | 374,968 |
| | 25 | 0,05 | 0,003 | 72,231 |
| | | 1 | $-4{,}594\times10^{-6}$ | 212,104 |
| | 50 | 0,05 | $1{,}978\times10^{-4}$ | 138,299 |
| | | 1 | $-1{,}122\times10^{-5}$ | 422,944 |
| | 100 | 0,05 | $-1{,}764\times10^{-6}$ | 325,925 |
| | | 1 | $-1{,}000\times10^{-5}$ | 824,361 |
| | 200 | 0,05 | -0,060 | 574,591 |
| | | 1 | -0,072 | 2095,373 |
| | 400 | 0,05 | -0,060 | 1399,845 |
| | | 1 | -0,068 | 4758,842 |

Berdasarkan Tabel 4.5, hasil yang baik dalam seluruh kondisi besar populasi dan besar peluang mutasi didapat ketika uji pada variabel sebesar 100 dan 1000. Namun, pada uji 10000 variabel, didapat hasil yang kurang baik dalam seluruh kondisi besar populasi dan besar peluang mutasi. Hal ini dikarenakan nilai *fitness* yang dihasilkan memiliki *error* melebihi 0,001. Sedangkan pada uji 100000 variabel, didapat hasil yang baik pada kondisi besar populasi 25, 50, dan 100, dengan kondisi besar peluang mutasi ketika besar populasi 25 sebesar 1.

### 4.3.6 Uji fungsi Quintic

Tabel 4.6: Hasil uji pada fungsi quintic

| Jumlah Variabel | Besar Populasi | Peluang Mutasi | Nilai Fitness | Waktu (detik) |
|---|---|---|---|---|
| 100 | 25 | 0,05 | 0,364 | 0,245 |
| | | 1 | 0,015 | 0,506 |
| | 50 | 0,05 | 0,039 | 0,448 |
| | | 1 | 0,003 | 0,807 |
| | 100 | 0,05 | 0,003 | 0,725 |
| | | 1 | $6{,}334\times10^{-4}$ | 1,484 |
| | 200 | 0,05 | $4{,}288\times10^{-4}$ | 1,596 |
| | | 1 | $3{,}397\times10^{-6}$ | 2,947 |
| | 400 | 0,05 | $2{,}111\times10^{-5}$ | 2,837 |
| | | 1 | $5{,}254\times10^{-8}$ | 5,68 |
| 1000 | 25 | 0,05 | 5,055 | 1,42 |
| | | 1 | 0,326 | 3,385 |
| | 50 | 0,05 | 1,432 | 2,823 |
| | | 1 | 0,070 | 6,504 |
| | 100 | 0,05 | 0,375 | 5,563 |
| | | 1 | 0,010 | 12,888 |
| | 200 | 0,05 | 0,35 | 11,756 |
| | | 1 | $2{,}790\times10^{-4}$ | 26,537 |
| | 400 | 0,05 | 0,03 | 24,301 |
| | | 1 | $7{,}730\times10^{-6}$ | 53,566 |
| 10000 | 25 | 0,05 | 56,650 | 12,896 |
| | | 1 | 3,837 | 31,345 |
| | 50 | 0,05 | 21,050 | 25,31 |
| | | 1 | 0,789 | 67,08 |
| | 100 | 0,05 | 4,903 | 51,678 |
| | | 1 | 0,125 | 131,48 |

Tabel 4.6 Hasil uji pada fungsi quintic (lanjutan)

| Jumlah Variabel | Besar Populasi | Peluang Mutasi | Nilai Fitness | Waktu (detik) |
|---|---|---|---|---|
| 100000 | 200 | 0,05 | 0,735 | 126,408 |
| | | 1 | 0,008 | 278,444 |
| | 400 | 0,05 | 0,068 | 242,553 |
| | | 1 | $2{,}783\times10^{-4}$ | 599,535 |
| | 25 | 0,05 | 668,894 | 130,143 |
| | | 1 | 36,606 | 300,263 |
| | 50 | 0,05 | 249,522 | 251,136 |
| | | 1 | 9,404 | 605,697 |
| | 100 | 0,05 | 49,308 | 585,407 |
| | | 1 | 1,415 | 1342,058 |
| | 200 | 0,05 | 8,978 | 1327,628 |
| | | 1 | 0,125 | 3143,324 |
| | 400 | 0,05 | 0,979 | 3197,62 |
| | | 1 | 0,004 | 7693,484 |

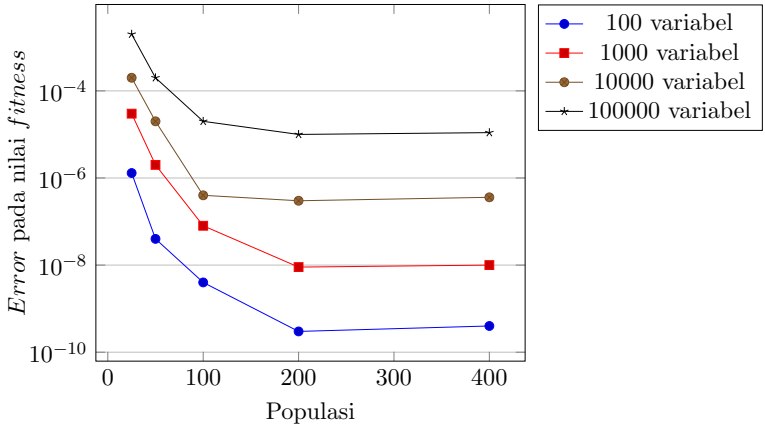
Dalam uji pada fungsi *quintic* berdasarkan Tabel 4.6, program yang telah disusun mendapatkan hasil yang baik pada uji 100, 1000, dan 10000 variabel dalam kondisi besar populasi dan besar peluang mutasi tertentu. Hal tersebut dikarenakan nilai *fitness* yang dihasilkan oleh program memiliki *error* lebih kecil dari 0,001. Pada uji 100 variabel, didapat hasil yang baik pada kondisi besar populasi 100, 200, dan 400, pada besar populasi 100, didapat hasil yang baik ketika besar peluang mutasi 1. Pada uji 1000 variabel, didapat hasil yang baik ketika kondisi besar populasi 200 dan 400 dengan besar peluang mutasi 1. Pada uji 10000 variabel, hanya didapat hasil yang baik ketika besar populasi 400 dan besar peluang mutasi sebesar 1.

## 4.4 Analisis Pengaruh Besar Populasi dan Besar Peluang Mutasi Terhadap Hasil Uji

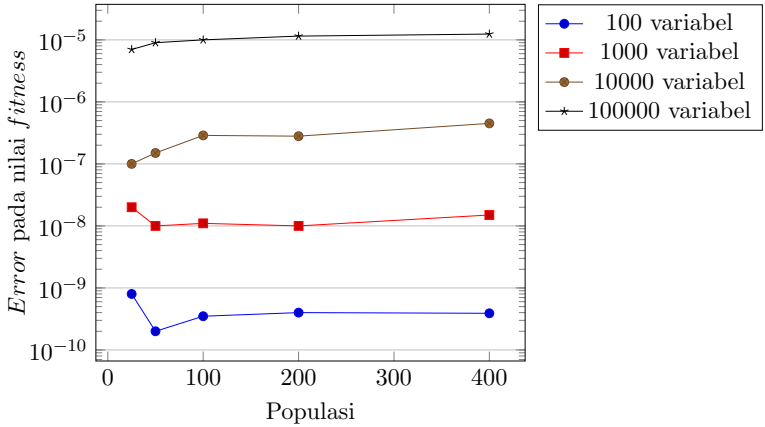
Hasil uji yang sudah didapat kemudian ditentukan pengaruh besar populasi terhadap nilai *fitness* yang dihasilkan untuk menentukan besar populasi yang ideal dalam menyelesaikan masalah optimasi fungsi dengan banyak variabel pada fungsi-fungsi yang diuji. Pengaruh besar populasi terhadap nilai *fitness* yang dihasilkan pada masing-masing uji fungsi akan ditampilkan dalam grafik dengan membandingkan *error* pada nilai *fitness* yang dihasilkan terhadap besar populasi.

### 4.4.1 Uji fungsi Sphere

Berdasarkan Gambar 4.2, didapat bahwa dengan uji ketika peluang mutasi sebesar 0,05, semakin besar populasi, didapat nilai *fitness* yang lebih mendekati nilai optimal. Namun hal tersebut hanya berlaku pada besar populasi 25, 50, 100, dan 200. Ketika besar populasi mencapai 400, tidak didapat hasil yang lebih baik dibandingkan nilai *fitness* ketika besar populasi 200. Dalam uji pada peluang mutasi 1, semakin besar populasi tidak menjamin didapatnya hasil yang lebih baik dibandingkan dengan populasi yang lebih rendah. Namun, hasil ketika peluang mutasi sebesar 1 selalu lebih baik dibandingkan dengan uji pada peluang mutasi sebesar 0,05. Sehingga dalam uji pada fungsi *sphere*, besar populasi 50 dan besar peluang mutasi 1 menjadi kondisi yang ideal dalam menyelesaikan masalah optimasi fungsi dengan uji pada fungsi tersebut.




(a) peluang mutasi 0,05

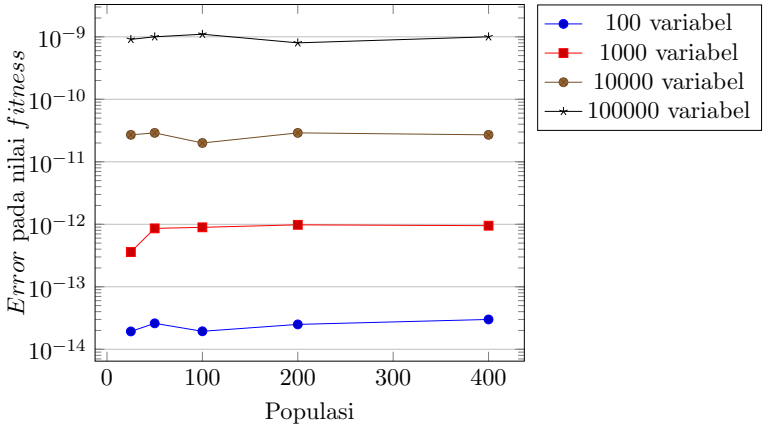



(b) peluang mutasi 1

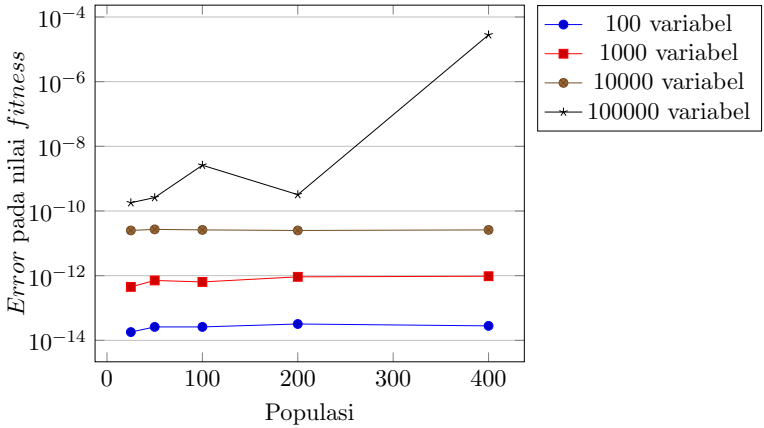
Gambar 4.2: Grafik nilai *fitness* terhadap besar populasi pada uji fungsi *Sphere*

### 4.4.2 Uji fungsi Csendes

Berdasarkan Gambar 4.3, dalam uji pada fungsi *csendes*, semakin besar populasi dan semakin besar peluang mutasi




(a) peluang mutasi 0,05



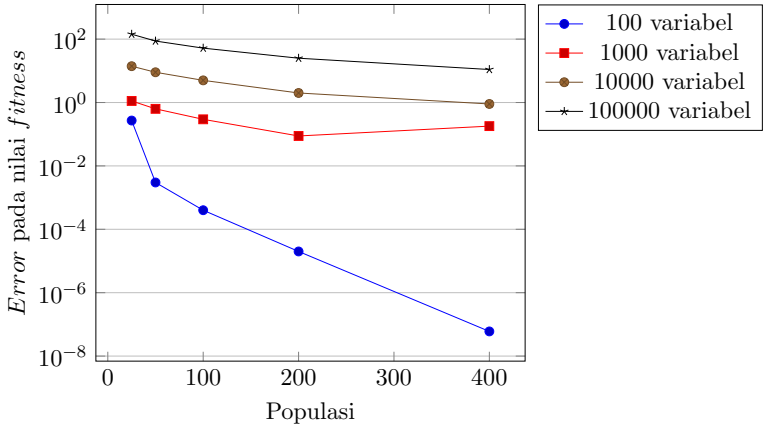

(b) peluang mutasi 1

Gambar 4.3: Grafik nilai *fitness* terhadap besar populasi pada uji fungsi *Csendes*

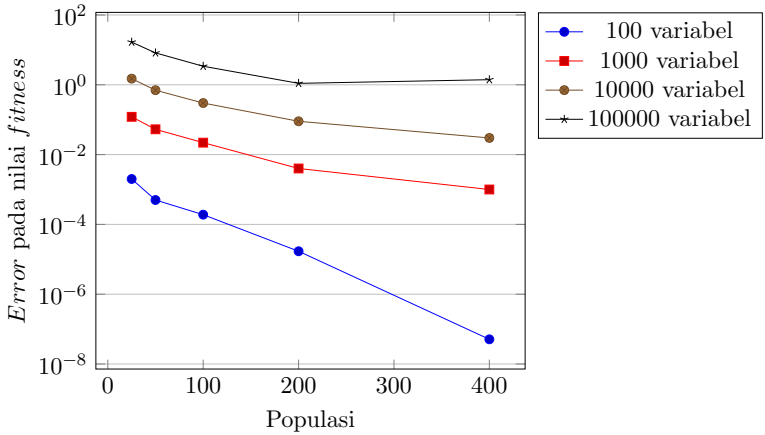
tidak menjamin hasil yang lebih baik dibandingkan dengan kondisi besar populasi dan peluang mutasi yang lebih rendah, dengan semua nilai *error* yang hampir sama, besar populasi

25 dan peluang mutasi 0,05 menjadi kondisi yang ideal untuk menyelesaikan masalah optimasi fungsi pada uji fungsi ini.

### 4.4.3 Uji fungsi Alpine




(a) peluang mutasi 0,05




(b) peluang mutasi 1

Gambar 4.4: Grafik nilai *fitness* terhadap besar populasi pada uji fungsi *Alpine*

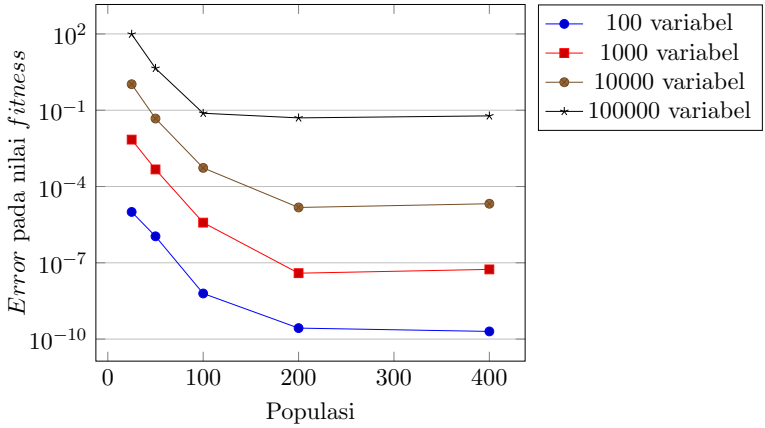
Berdasarkan Gambar 4.4, dalam uji pada fungsi *alpine*, semakin besar populasi dan semakin besar peluang mutasi, didapat hasil yang lebih baik.

### 4.4.4 Uji fungsi Sum Squares

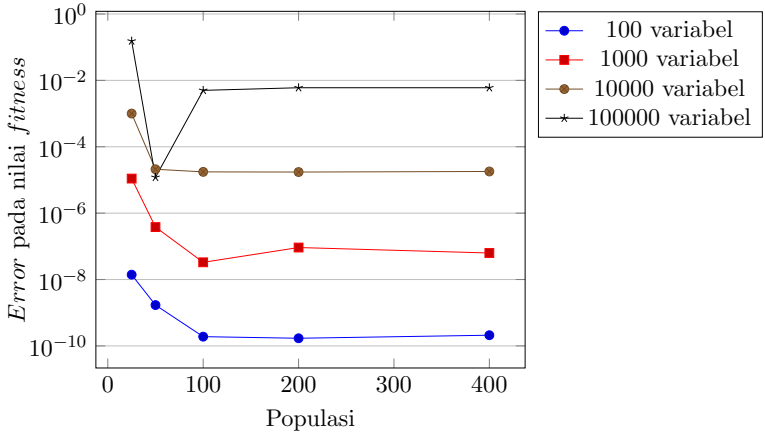
Berdasarkan Gambar 4.5, dalam uji pada fungsi *sumsquares*, didapat bahwa dengan uji ketika peluang mutasi sebesar 0,05, semakin besar populasi, didapat nilai *fitness* yang lebih mendekati nilai optimal. Namun hal tersebut hanya berlaku pada besar populasi 25, 50, 100, dan 200. Ketika besar populasi mencapai 400, tidak didapat hasil yang lebih baik dibandingkan nilai *fitness* ketika besar populasi 200. Dalam uji pada peluang mutasi 1, semakin besar populasi tidak menjamin didapatnya hasil yang lebih baik dibandingkan dengan populasi yang lebih rendah. Namun, hasil ketika peluang mutasi sebesar 1 selalu lebih baik dibandingkan dengan uji pada peluang mutasi sebesar 0,05. Sehingga dalam uji pada fungsi *sum squares*, besar populasi 50 dan besar peluang mutasi 1 menjadi kondisi yang ideal dalam menyelesaikan masalah optimasi fungsi dengan uji pada fungsi tersebut.

### 4.4.5 Uji fungsi Schumer-Steiglitz

Berdasarkan Gambar 4.6, dalam uji pada fungsi *schumer-steiglitz*, semakin besar populasi dan semakin besar peluang mutasi tidak menjamin hasil yang lebih baik dibandingkan dengan kondisi besar populasi dan peluang mutasi yang lebih rendah, dengan semua nilai *error* yang hampir sama, besar populasi 100 dan peluang mutasi 0,05 menjadi kondisi yang ideal untuk menyelesaikan masalah optimasi fungsi pada uji fungsi ini.




(a) peluang mutasi 0,05

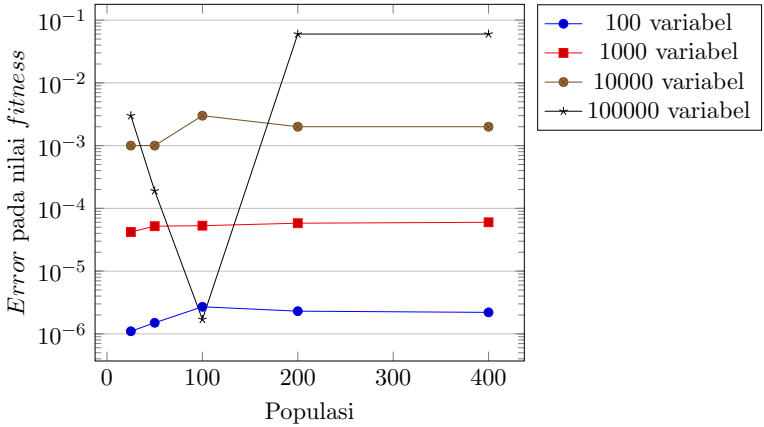



(b) peluang mutasi 1

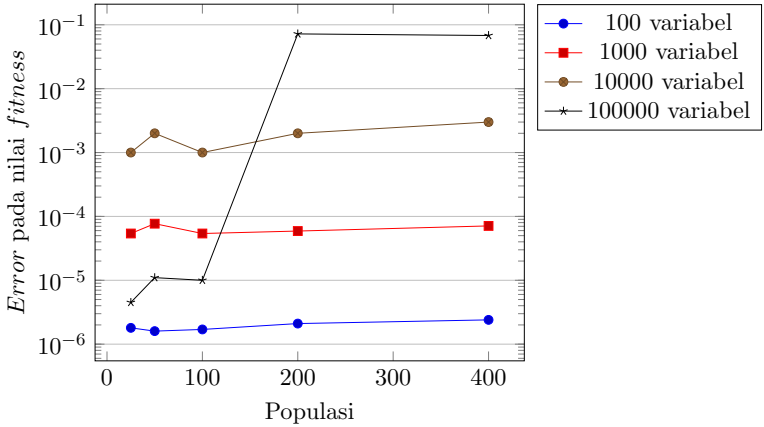
Gambar 4.5: Grafik nilai *fitness* terhadap besar populasi pada uji fungsi *SumSquares*

### 4.4.6 Uji fungsi Quintic

Berdasarkan Gambar 4.7, dalam uji pada fungsi *quintic*, semakin besar populasi dan semakin besar peluang mutasi,



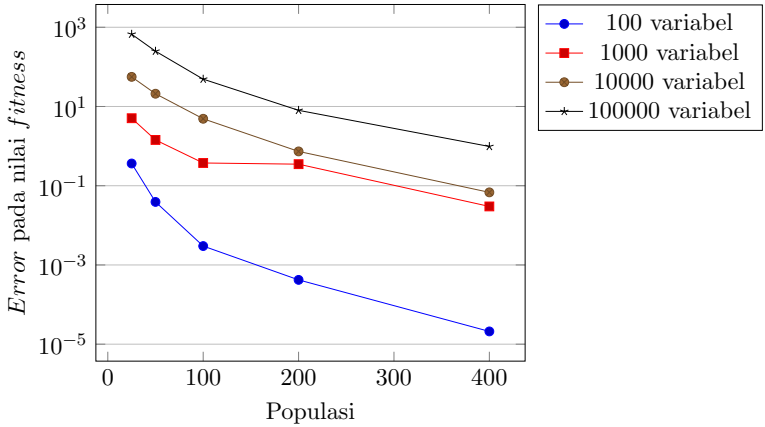

(a) peluang mutasi 0,05



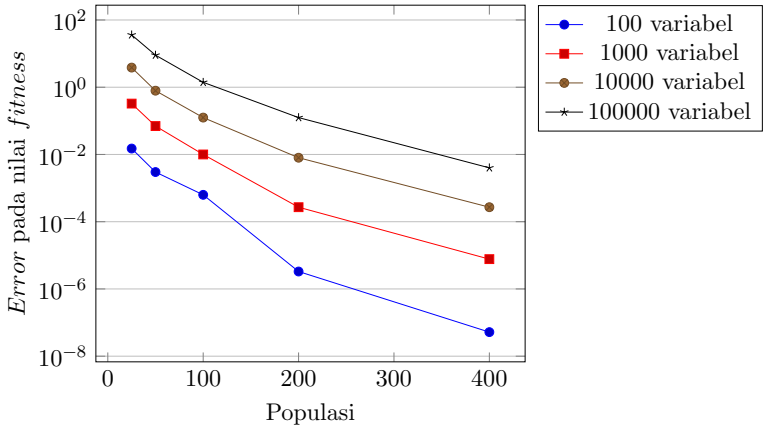

(b) peluang mutasi 1

Gambar 4.6: Grafik nilai *fitness* terhadap besar populasi pada uji fungsi $Schumer - Steiglitz$

didapat hasil yang lebih baik.




(a) peluang mutasi 0,05




(b) peluang mutasi 1

Gambar 4.7: Grafik nilai *fitness* terhadap besar populasi pada uji fungsi *Quintic*

### 4.4.7 Analisis Kasus Tidak Berpengaruhnya Perbedaan Besar Populasi Pada Uji Fungsi Tertentu

Berdasarkan hasil uji diatas, didapat bahwa besar jumlah populasi berpengaruh terhadap hasil dari algoritma genetika

yang telah dirumuskan pada beberapa uji fungsi tertentu, yaitu fungsi *Alpine*, fungsi *Quintic*, fungsi *Sphere*, dan fungsi *Sum Squares*. Namun pada fungsi *Sphere* dan fungsi *Sum Squares*, hanya berpengaruh ketika peluang mutasi sebesar 0,05 dan ketika besar populasi hanya mencapai 100. Dalam uji pada fungsi-fungsi tersebut, semakin besar jumlah populasi, nilai *fitness* yang dihasilkan semakin mendekati nilai global optimum yang seharusnya. Pada uji fungsi *Alpine* dan fungsi *Quintic*, kondisi mutasi terjadi pada seluruh kromosom menghasilkan nilai *fitness* yang lebih baik dibandingkan dengan kondisi peluang mutasi sebesar 0,05. Artinya, uji pada kedua fungsi tersebut memiliki hasil yang paling baik ketika besar populasi sebesar 400 dan peluang mutasi sebesar satu atau terjadi pada seluruh kromosom. Perbedaan besar populasi antara 25, 50, dan 100 tidak berpengaruh pada fungsi *Csendes* dan fungsi *Schumer – Steiglitz*, pada fungsi *Sphere* dan fungsi *Sum Squares*, tidak berpengaruh ketika peluang mutasi sebesar satu. Hal ini dikarenakan ketika algoritma genetika yang telah dirumuskan diimplementasikan kedalam bahasa pemrograman, tipe bilangan yang digunakan adalah bilangan *double* atau *floating-point numbers*. Bilangan ini biasanya kurang akurat karena menggunakan bilangan terbatas dalam merepresentasikan bilangan asli. Konsekuensi dari kurang akuratnya bilangan ini adalah ketika pembulatan nilai error, ketika melakukan operasi penjumlahan atau pengurangan nilai antara bilangan kecil dan bilangan besar, beberapa digit dalam bilangan kecil akan dibulatkan karena digit yang terbatas dalam merepresentasikan hasil operasi tersebut. Kesalahan dalam pembulatan saat perhitungan ini yang menyebabkan ketidakakuratan yang signifikan terhadap hasil operasi[18]. Dalam penelitian ini, hal tersebut berkaitan dengan proses perhitungan nilai *fitness* ketika proses *crossover* dan mutasi. Sudah dilakukan usaha

dalam meminimalisir kesalahan dalam perhitungan. Yaitu menggunakan algoritma penjumlahan *neumaier*. Namun hal tersebut tidak berpengaruh pada fungsi *csendes* dan *schumer-steiglitz*, kedua fungsi tersebut memiliki pangkat yang cukup besar, ketika suatu solusi sudah mulai mendekati nol atau minimumnya, nilai $fitness$ memiliki nilai yang sangat kecil, algoritma penjumlahan yang digunakan tidak mampu menghasilkan akurasi yang akurat ketika menghitung nilai $fitness$. Hal ini yang menyebabkan tidak berpengaruhnya besar populasi terhadap hasil. Terkait dengan tidak berpengaruhnya besar populasi terhadap fungsi *sphere* dan *sum squares* ketika peluang mutasi sebesar satu, hal ini berkaitan dengan penjelasan diatas yang mengatakan bahwa operasi pada *double* kurang akurat karena menggunakan bilangan terbatas dalam merepresentasikan bilangan asli, hal tersebut mempengaruhi hasil operasi bilangan yang besar dengan bilangan kecil. Ketika mutasi terjadi pada seluruh kromosom atau peluang mutasi sebesar satu, selalu dilakukan perubahan nilai pada kromosom dengan cara menjumlahkan bilangan antara -0,5 sampai 0,5. Dengan kata lain, ketika solusi sudah mulai mendekati nol, terjadi operasi antara bilangan yang besar dengan bilangan kecil, ketika operasi tersebut, bilangan kecil akan dibulatkan karena digit yang terbatas dalam merepresentasikan hasil operasi, sehingga memiliki hasil yang kurang akurat. Kesalahan kecil dalam tiap generasi, akan menghasilkan kesalahan yang besar ketika generasi sudah mencapai ratusan bahkan 1000. Kesalahan perhitungan tersebut tidak berpengaruh pada fungsi Alpine dan Quintic, karena solusi optimum dari kedua fungsi tersebut tidak mendekati nol. Dalam penelitian ini, hanya dilakukan uji dengan besar populasi mencapai 400 saja, hal ini dikarenakan ketika algoritma yang telah dirumuskan diimplementasikan dalam bahasa pemrograman JAVA dan

diuji dengan jumlah variabel 100000 dan besar populasi diatas 400, menghasilkan keluaran $java.lang.OutOfMemoryError$ yang berarti uji pada kondisi tersebut membutuhkan memori dari kapasitas yang tersedia.

Terkait dengan kurang baiknya nilai $fitness$ yang dihasilkan pada uji fungsi *Alpine* dan fungsi *Quintic* pada uji 100000 variabel, ketika solusi dalam kromosom dievaluasi kedalam fungsi $fitness$ atau ketika dilakukan perhitungan $f(x)$ pada $x$, nilai error pada nilai $fitness$ yang dihasilkan tidak lebih kecil dari nilai error pada solusi yang dihasilkan. Sebagai contoh jika dilakukan perhitungan pada fungsi *Quintic* pada 2.4, fungsi tersebut memiliki solusi yang terletak pada -1, atau -0,4026, atau 2. Misal, jika solusi yang dihasilkan pada algoritma genetika yang telah dirumuskan adalah 2,0001 atau memiliki error sebesar 0,0001, maka nilai $fitness$ yang dihasilkan sebesar 0,00300034, sehingga jika dalam 1000 variabel keatas, solusi yang dimiliki harus memiliki error yang sangat kecil agar didapat nilai $fitness$ yang baik. Hal ini juga berlaku pada uji fungsi *Alpine*.

# BAB V
# PENUTUP

## 5.1 Kesimpulan

1. Dalam perumusan algoritma genetika agar mampu menyelesaikan optimasi fungsi dengan banyak variabel, dirumuskan representasi kromosom yang berisi solusi optimal dari fungsi yang diuji dengan nilai awal bilangan acak dengan rentang sesuai dengan tetapan pada fungsi yang diuji, dengan operator *crossover* adalah *Average Crossover* dan operator seleksi yaitu *Tournament Selection*. Fungsi $fitness$ dalam algoritma genetika yang telah dirumuskan adalah fungsi yang diuji, pada penelitian ini perhitungan nilai *fitness* dilakukan dalam proses mutasi dan *crossover*. sehingga setelah melewati kedua proses tersebut, tidak perlu menghitung nilai *fitness* karena sudah mempunyai nilai *fitness* baru.

2. Algoritma genetika yang telah dirumuskan mendapatkan hasil akurasi yang baik dalam menyelesaikan optimasi fungsi dengan banyak variabel pada uji fungsi *Sphere*, fungsi *Csendes*, fungsi *Sum Squares*, fungsi *Schumer-Steiglitz*, dan fungsi *Quintic*. Namun pada uji fungsi *Schumer-Steiglitz*, hanya didapat hasil yang baik pada uji 100, 1000, 100000 variabel dan pada uji fungsi *Quintic*, tidak didapat hasil yang baik pada uji 100000 variabel.

3. Dalam algoritma genetika yang telah dirumuskan, peluang mutasi 1 atau mutasi terjadi pada seluruh kromosom menghasilkan hasil yang lebih baik jika

dibandingkan dengan besar peluang mutasi 0,05 pada uji fungsi *Sphere*, *Alpine*, *Sum Squares*, *Schumer-Steiglitz*, dan *Quintic*.

4. Dalam algoritma genetika yang telah dirumuskan, semakin besar jumlah populasi tidak menjamin didapat hasil yang lebih baik. Hanya uji pada fungsi *Quintic* dan fungsi *Alpine* yang didapat hasil yang lebih baik jika populasi semakin besar.

## 5.2 Saran

Saran dari penulis untuk penelitian selanjutnya adalah:

1. Algoritma genetika yang sudah dirumuskan pada penelitian ini dapat dikembangkan untuk bisa menyelesaikan optimasi fungsi *Alpine*, *Quintic*, dan fungsi-fungsi lain dengan lebih baik.

2. Penelitian ini dapat dikembangkan untuk menyelesaikan optimasi dengan jumlah variabel yang lebih banyak

# DAFTAR PUSTAKA


[1] Yang, Xin-She.(2014), **Nature-Inspired Optimization Algorithms**, Elsevier, London.

[2] Yang, S., Yang, M., Wang, S. and Huang, K., 2016. Adaptive immune genetic algorithm for weapon system portfolio optimization in military big data environment. **Cluster Computing**, 19(3), hal.1359-1372.

[3] Arora, Rajesh Kumar., 2015. **Optimization Algorithms and Applications**, CRC Press.

[4] Indrianingsih, Y., 2010. Algoritma Genetik Untuk Menyelesaikan Masalah Optimasi Fungsi Berkendala Dengan Pengkodean Bilangan Bulat. **Jurnal Angkasa Sekolah Tinggi Teknologi Adisutjipto (STTA)**, 2(1), hal.067-076.

[5] Yang, X., 2008. **Introduction to mathematical optimization. From Linear Programming to Metaheuristics.**

[6] Rumahorbo, R.L. and Mansyur, A., 2016. Konsistensi metode simpleks dalam menentukan nilai optimum. **KARISMATIKA: Kumpulan Artikel Ilmiah, Informatika, Statistik, Matematika dan Aplikasi**, 3(1).

[7] James, J.Q. and Li, V.O., 2015. A social spider algorithm for global optimization. **Applied Soft Computing**, 30, hal.614-627.

[8] Sun, Y., Wang, X., Chen, Y. and Liu, Z., 2018. A modified whale optimization algorithm for large-scale global optimization problems. **Expert Systems with Applications**, 114, hal.563-577.

[9] Mahmudy, W.F., 2008. Optimasi fungsi tanpa kendala menggunakan algoritma genetika dengan kromosom biner dan perbaikan kromosom hill-climbing. **Kursor**, 4(1), hal.23-29.

[10] Mahmudy, W.F., 2009, February. Optimasi Fungsi Tak Berkendala Menggunakan Algoritma Genetika Terdistribusi dengan Pengkodean Real. **Seminar Nasional Basic Science VI**, Universitas Brawijaya, Malang, hal.1-5.

[11] Mahardhika, L.T., 2015. Algoritma Genetika yang Diautomasi untuk Optimasi Fungsi Uji Dejong dan Ackley. **Likhitaprajna**, 17(1), hal.97-108.

[12] Yongshou, D., Yuanyuan, L., Lei, W., Junling, W. and Deling, Z., 2007. Adaptive immune-genetic algorithm for global optimization to multivariable function. **Journal of Systems Engineering and Electronics**, 18(3), hal.655-660.

[13] Ali, A.F. and Tawhid, M.A., 2017. A hybrid particle swarm optimization and genetic algorithm with population partitioning for large scale optimization problems. **Ain Shams Engineering Journal**, 8(2), hal.191-206.

[14] Zhang, Y., Zhou, M., Jiang, Z. and Liu, J., 2015, May. A multi-agent genetic algorithm for big optimization problems. **2015 IEEE Congress on Evolutionary Computation (CEC)** hal. 703-707.

[15] Coley, D.A., 1999. **An introduction to genetic algorithms for scientists and engineers**. World Scientific Publishing Company.

[16] Umbarkar, A.J. dan Sheth, P.D., 2015. Crossover operators in genetic algorithms: a review. **ICTACT journal on soft computing**, 6(1).

[17] Neumaier, A., 1973. **Rundungsfehleranalyse einiger verfahren zur summation endlicher summen**. Univ., Inst. f. Prakt. Mathematik.

[18] Zou, D., Wang, R., Xiong, Y., Zhang, L., Su, Z. and Mei, H., 2015, May. A genetic algorithm for detecting significant floating-point inaccuracies. **2015 IEEE/ACM 37th IEEE International Conference on Software Engineering** Vol. 1, hal. 529-539.

*Halaman ini sengaja dikosongkan*

# LAMPIRAN

*Halaman ini sengaja dikosongkan*

# LAMPIRAN 1

Library.java

```
package finalproject;
//Library class for store the variables | kelas Library untuk menympan jumlah variabel
public class Library {
    private static int numOfVariables;
    public static void setNumOfVariables(int a){
        numOfVariables = a;
    }
    public static int getNumOfVariables(){
        return numOfVariables;
    }
    public static double neumaierSum(double... fa){
        double sum = 0.0;
        double c = 0.0;
        for(int i = 0; i < fa.length; i++){
            double t = sum + fa[i];
            if(Math.abs(sum) >= Math.abs(fa[i])) {
                c += (sum - t) + fa[i];
            }else{
                c += (fa[i] - t) + sum;
            }
            sum = t;
        }
        return sum + c;
    }
}
```

# LAMPIRAN 2

Solution.java

```
/*
Solution class for generate, set, and get chromosome's value
kelas solusi untuk men-generate, set, dan mendapatkan nilai/isi kromosom
*/
package finalproject;

public class Solution {
    //inisiasi array x dan nilai fitness
    double x[] = new double[Library.getNumOfVariables()];
    double fitness = 0;
    //constructor, generate x dengan bilangan random
    public Solution(){
        for(int i = 0; i < x.length; i++){
            //range bilangan sesuai dengan batasan dalam fungsi
            //1. Sphere
            x[i] = Math.random()*200-100;
            //2. Csendes
            //x[i] = Math.random()*2-1;
            //3. Alpine
            //x[i] = Math.random()*20-10;
            //4. Sum Squares
            //x[i] = Math.random()*20-10;
            //5. Schumer Steiglitz
            //x[i] = Math.random()*200-100;
            //6. Quintic
            //x[i] = Math.random()*20-10;
```

```
        }
    }

    //inisiasi kromosom baru dengan mendapatkan kromosom lain
    //constructor, generate x sesuai dengan array a
    public Solution(double[] a, double f){
        for(int i = 0; i < x.length; i++){
            x[i] = a[i];
        }
        fitness = f;
    }

    //function untuk mengubah isi kromosom
    //hitung nilai fitness ketika mengubah isi
    public void setX(int i, double a){
        double b = x[i];
        x[i]=a;
        double[] storage = new double[3];
        storage[0] = fitness;
        //1. Sphere

        storage[1] = a*a;
        storage[2] = -1*b*b;

        //2. Csendes
        /*
        storage[1] = a*a*a*a*a*a*(2 + Math.sin(1/a));
        storage[2] = -1*b*b*b*b*b*b*(2 + Math.sin(1/b));
        */
        //3. Alpine
        /*
        storage[1] = Math.abs(a*Math.sin(a) + 0.1*a);
        storage[2] = -1*Math.abs(b*Math.sin(b) +
```

```
            0.1*b);
        */
        //4. Sum Squares
        /*
        //storage[1] = i*Math.pow(a, 2);
        //storage[2] = -1*i*Math.pow(b, 2);
        storage[1] = i*a*a;
        storage[2] = -1*(i*b*b);
        */
        //5. Schumer Steiglitz
        /*
        storage[1] = a*a*a*a;
        storage[2] = -1*(b*b*b*b);
        */
        //6. Quintic
        /*
        storage[1] = Math.abs(a*a*a*a*a - 3*a*a*a*a + 4*a*a*a + 2*a*a - 10*a - 4);
        storage[2] = -1*Math.abs(b*b*b*b*b - 3*b*b*b*b + 4*b*b*b + 2*b*b - 10*b - 4);
        */
        fitness = Library.neumaierSum(storage);
    }

    //function untuk mendapatkan kromosom
    public double[] getX(){
        return x;
    }

    //function untuk mendapatkan nilai tertentu dalam kromosom
    public double getX(int i){
        return x[i];
    }

    //function untuk evaluasi fungsi
    public double f(){
        double result = 0;
```

```
        //1. Sphere

        for(int i = 0; i < x.length; i++){
            result += x[i]*x[i];
        }
        //2. Csendes
        /*
        for(int i = 0; i< x.length; i++){
            result += x[i]*x[i]*x[i]*x[i]*x[i]*x[i]*(2 + Math.sin(1/x[i]));
        }*/
        //3. Alpine
        //(-9.324, -6.383, -3.041, 0.1, 0, 3.24, 6.183, 9.525)
        /*
        for(int i = 0; i< x.length; i++){
            result += Math.abs(x[i]*Math.sin(x[i]) + 0.1*x[i]);
        }*/
        //4. Sum Squares
        /*
        for(int i = 0; i < x.length; i++) {
            result += i*(x[i]*x[i]);
        }*/
        //5. Schumer Steiglitz
        /*
        for(int i = 0; i < x.length; i++) {
            result += x[i]*x[i]*x[i]*x[i];
        }*/
        //6. Quintic
        // (-1,-0.4026,2)
        /*
        for(int i = 0; i < x.length; i++) {
            result += Math.abs(x[i]*x[i]*x[i]*x[i]*x[i] - 3*x[i]*x[i]*x[i]*x[i] + 4*x[i]*x[i]*x[i] + 2*x[i]*x[i] - 10*x[i] - 4);
        }*/
```

```
        return result;
    }

    //function untuk mendapatkan nilai fitness
    //jika fitness nol, maka dibawa ke fungsi
       evaluasi
    public double getFitness(){
        if(fitness==0){
            fitness = f();
        }
        return fitness;
    }

    //function untuk print isi kromosom
    public void printX(){
        for(int i = 0; i < x.length; i++){
            System.out.println(i + "\t" + x[i]);
        }
        System.out.println();
    }

    //function untuk print nilai fitness
    public void printFitness(){
        System.out.println("fitness\t" +
           getFitness());
    }
}
```

# LAMPIRAN 3

Population.java

```
/*
Population class for store and generate the solutions, and find the fittest from all solutions
kelas populasi untuk men-generate dan menympan solusi, serta mendapatkan fittest dari semua solusi yang ada
*/
package finalproject;
public class Population {
    Solution[] solution;

    //Constructor Population
    public Population(int numberOfSolutions, boolean initialization){
        solution = new Solution[numberOfSolutions];
        //jika inisiasi, generate populasi berisi kromosom
        if(initialization){
            for(int i = 0; i<numberOfSolutions; i++){
                Solution newSolution = new Solution();
                saveSolution(i,newSolution);
            }
        }
    }

    //function untuk menyimpan kromosom kedalam populasi
```

```
    public void saveSolution(int index, Solution newSolution){
        solution[index] = newSolution;
    }

    //function untuk mendapatkan kromosom
    public Solution getSolution(int index){
        return solution[index];
    }

    //function untuk mendapatkan kromosom terbaik dalam populasi
    public Solution getFittest(){
        Solution fittest = solution[0];
        for(int i = 0; i<solution.length; i++){
            if(getSolution(i).getFitness()<fittest.getFitness()){
                fittest = getSolution(i);
            }
        }
        return fittest;
    }

    public int popSize(){
        return solution.length;
    }

}
```

# LAMPIRAN 4

GeneticAlgorithm.java

```
/*
this class contains crossover. mutation, selection, and evolve methods
*/
package finalproject;
public class GeneticAlgorithm {
    private static final double mutationrate = 0.05;
    private static final int tournamentsize = 5;
    private static final boolean elitism = true;

    //crossover
    public static Solution crossover(Solution parent1, Solution parent2){
        //membuat kromosom anak, ambil dari parent1
        Solution child1 = new Solution(parent1.getX(), parent1.getFitness());
        for(int i = 0; i < Library.getNumOfVariables(); i++){
            //simpan nilai fitness sebelum crossover untuk dibandingkan
            double f = child1.getFitness();
            //ubah isi kromosom ke-i
            child1.setX(i, (parent1.getX(i) + parent2.getX(i))/2);
            //jika lebih buruk, ubah seperti semula
            if(child1.getFitness() > f){
                child1.setX(i, parent1.getX(i));
            }
```

```
        }
        return child1;
    }
    //mutasi
    public static void mutation(Solution solution){
        //membuat kromosom dari kromosom yang akan dimutasi, untuk mengembalikan nilai jika lebih buruk
        Solution solution2 = new Solution(solution.getX(), solution.getFitness());
        for(int i = 0; i < Library.getNumOfVariables(); i++){
            //simpan nilai fitness sebelum mutasi untuk dibandingkan
            double f = solution.getFitness();
            //ubah isi kromosom ke-i
            solution.setX(i, solution.getX(i)+Math.random()-0.5);
            //jika lebih buruk, ubah seperti semula
            if(solution.getFitness() > f){
                solution.setX(i, solution2.getX(i));
            }
        }
    }
    //seleksi turnamen
    public static Solution tournament(Population pop){
        //geneate populasi baru sebanyak tournamentsize
        Population tournamentselection = new Population(tournamentsize, false);
        for(int i = 0; i < tournamentsize; i++){
            //declare random untuk mendapatkan nilai acak untuk index
```

```
                getSolution
268             int random = (int)(Math.random()*pop
                .popSize());
269             //simpan kromosom pada index random
                ke populasi tournamentselection
270             tournamentselection.saveSolution(i,
                pop.getSolution(random));
271         }
272         //dapatkan nilai fittest
273         Solution fittest = tournamentselection.
            getFittest();
274         return fittest;
275     }
276     //evolusi
277     public static Population evolve(Population
        pop){
278         //generate populasi baru sebanyak jumlah
             populasi
279         //akan dipakai untuk menyimpan hasil
            proses evolusi
280         Population newpopulation = new
            Population(pop.popSize(), false);
281         int elitismoffset = 0;
282         //elitisim true-> simpan kromosom
            terbaik dari populasi yang akan
            dievolusi ke populasi baru
283         if(elitism){
284             newpopulation.saveSolution(0, pop.
                getFittest());
285             elitismoffset = 1;
286         }
287         //lakukan crossover ke seluruh kromosom
288         for(int i = elitismoffset; i < pop.
            popSize(); i++){
289             //parent1 diambil dari kromosom ke-i
290             Solution parent1 = pop.getSolution(i
                );
291             //parent2 diambil dari seleksi
```

```
                turnamen
            Solution parent2 = tournament(pop);
            //crossover
            Solution child = crossover(parent1,
                parent2);
            //simpan hasil kromosom
            newpopulation.saveSolution(i, child)
                ;
        }
        //lakukan mutasi ke seluruh kromosom
            pada populasi baru
        for(int i = elitismoffset; i < pop.
            popSize(); i++){
            if(Math.random() < mutationrate){
                mutation(newpopulation.
                    getSolution(i));
            }
        }
        return newpopulation;
    }
}
```

# LAMPIRAN 5

Main.java

```
/*
Main class
*/
package finalproject;

public class Main {

    /**
    * @param args the command line arguments
    */
    public static void main(String[] args) {
        //mengukur waktu
        long start = System.currentTimeMillis();
        //jumlah variabel
        int numOfVariables = 100000;
        Library.setNumOfVariables(numOfVariables);
        //jumlah populasi
        int numOfPopulations = 50;
        //jumlah iterasi
        int numOfIterations = 1000;
        //inisiasi populasi baru
        Population pop = new Population(numOfPopulations, true);
        for(int i = 0; i < numOfIterations; i++){
            //lakukan algoritma genetika ke pop sebanyak jumlah iterasi
            pop = GeneticAlgorithm.evolve(pop);
            if(pop.getFittest().getFitness() <= 0.001){
```

```
                    break;
                }
            }
            //print isi kromosom terbaik
            pop.getFittest().printX();
            pop.getFittest().printFitness();
            long end = System.currentTimeMillis();
            float sec = (end - start) / 1000F;
            System.out.println("Dibutuhkan waktu "+
                sec+" detik");
        }
}
```

# BIODATA PENULIS

Penulis memiliki nama lengkap Fityan Azizi, lahir di Jakarta, 4 Juni 1998. Penulis menempuh pendidikan dasar di SDIT Iqro' Bekasi, SMPI Nurul Fikri Boarding School Serang, dan SMAIT Nurul Fikri Depok. Penulis melanjutkan pendidikan tingginya di Institut Teknologi Sepuluh Nopember di Departemen Matematika, selama menempuh pendidikan tinggi di Departemen Matematika ITS, penulis aktif di organisasi kemahasiswaan sebagai *staff Internal Affair* periode 2017-2018 dan 2018-2019. Di tahun 2018-2019 penulis juga tercatat sebagai *Steering Comitee* Padamu Himatika '18. Kritik dan Saran dapat dikirimkan melalui email ftyns57@gmail.com