

TUGAS AKHIR - IT184802

CLOUD PROVISIONING MENGGUNAKAN GENETIC ALGORITHM DAN ARTIFICIAL NEURAL NETWORK

CLOUD PROVISIONING USING GENETIC ALGORITHM AND ARTIFICIAL NEURAL NETWORK

BRYAN YEHUDA MANNUEL
NRP 05311940000021

Dosen Pembimbing I
Henning Titi Ciptaningtyas, S.Kom, M.Kom.
NIP 19840708 201012 2 004

Dosen Pembimbing II
Ridho Rahman Hariadi, S.Kom, M.Sc
NIP 19870213 201404 1 001

DEPARTEMEN TEKNOLOGI INFORMASI
Fakultas Teknologi Elektro dan Informatika Cerdas
Institut Teknologi Sepuluh Nopember
Surabaya
2022



TUGAS AKHIR - IT184802

CLOUD PROVISIONING MENGGUNAKAN GENETIC ALGORITHM DAN ARTIFICIAL NEURAL NETWORK

BRYAN YEHUDA MANNUEL
NRP 05311940000021

Dosen Pembimbing I
Henning Titi Ciptaningtyas, S.Kom, M.Kom.
NIP 19840708 201012 2 004

Dosen Pembimbing II
Ridho Rahman Hariadi, S.Kom, M.Sc
NIP 19870213 201404 1 001

DEPARTEMEN TEKNOLOGI INFORMASI
Fakultas Teknologi Elektro dan Informatika Cerdas
Institut Teknologi Sepuluh Nopember
Surabaya
2022



FINAL PROJECT - IT184802

CLOUD PROVISIONING USING GENETIC ALGORITHM AND ARTIFICIAL NEURAL NETWORK

BRYAN YEHUDA MANNUEL
NRP 05311940000021

Supervisor I
Henning Titi Ciptaningtyas, S.Kom, M.Kom.
NIP 19840708 201012 2 004

Supervisor II
Ridho Rahman Hariadi, S.Kom, M.Sc
NIP 19870213 201404 1 001

DEPARTEMEN OF INFORMATION TECHNOLOGY
Faculty of Intelligent Electrical and Informatics Technology
Institut Teknologi Sepuluh Nopember
Surabaya
2022

CLOUD PROVISIONING MENGGUNAKAN GENETIC ALGORITHM DAN ARTIFICIAL NEURAL NETWORK

Diajukan Untuk Memenuhi Salah Satu Syarat
Memperoleh Gelar Sarjana Komputer pada
Program Studi S-1 Departemen Teknologi Informasi
Departemen Teknologi Informasi
Fakultas Teknologi Elektro dan Informatika Cerdas
Institut Teknologi Sepuluh Nopember

NRP : 05311940000021

- Henning Titi Ciptaningtyas, S.Kom, M. Kom.
NIP. 19840708 201012 2 004
(Pembimbing 1)
- Ridho Rahman Hariadi, S.Kom, M. Sc.
NIP. 19870213 201404 1 001
(Pembimbing 2)
- xx.
NIP. xxxxxxxxxxxxxxxx
(Penguji 1)
- xx.
NIP. xxxxxxxxxxxxxxxx
(Penguji 2)

Desember 2022

LEMBAR PENGESAHAN

CLOUD PROVISIONING MENGGUNAKAN GENETIC ALGORITHM DAN ARTIFICIAL NEURAL NETWORK

TUGAS AKHIR

Diajukan Untuk Memenuhi Salah Satu Syarat
Memperoleh Gelar Sarjana Komputer pada
Program Studi S-1 Departemen Teknologi Informasi
Departemen Teknologi Informasi
Fakultas Teknologi Elektro dan Informatika Cerdas
Institut Teknologi Sepuluh Nopember

Oleh : **Bryan Yehuda Mannuel**

NRP : **05311940000021**

Disetujui oleh Tim Penguji Tugas Akhir :

Surabaya, 25 Desember 2022

Disetujui oleh

**KEPALA DEPARTEMEN TEKNOLOGI
INFORMASI**

SURABAYA

Desember 2022

CLOUD PROVISIONING USING GENETIC ALGORITHM AND ARTIFICIAL NEURAL NETWORK

Submitted to fulfill one of the requirements
for obtaining a degree Sarjana Komputer at
Undergraduate Study Program of Department of Information Technology
Department of Information Technology
Faculty of Intelligent Electrical and Informatics Technology
Institut Teknologi Sepuluh Nopember

NRP : 05311940000021

1. Henning Titi Ciptaningtyas, S.Kom, M. Kom.
NIP. 19840708 201012 2 004
(Supervisor 1)
2. Ridho Rahman Hariadi, S.Kom, M. Sc.
NIP. 19870213 201404 1 001
(Supervisor 2)
5. xxx.
NIP. xxxxxxxxxxxxxxxx
(Examiner 1)
6. xxx.
NIP. xxxxxxxxxxxxxxxx
(Examiner 2)

December 2022

APPROVAL SHEET

CLOUD PROVISIONING USING GENETIC ALGORITHM AND ARTIFICIAL NEURAL NETWORK

FINAL PROJECT

Submitted to fulfill one of the requirements
for obtaining a degree Sarjana Komputer at
Undergraduate Study Program of Department of Information Technology
Department of Information Technology
Faculty of Intelligent Electrical and Informatics Technology
Institut Teknologi Sepuluh Nopember

By : Bryan Yehuda Mannuel

NRP : 05311940000021

Approved by Final Project Examiner Team :

Surabaya, 25 December 2022

Approved by :

**HEAD OF DEPARTMENT OF INFORMATION
TECHNOLOGY**

SURABAYA

Desember 2022

PERNYATAAN ORISINALITAS

Yang bertanda tangan di bawah ini:

Nama Mahasiswa / NRP : Bryan Yehuda Mannuel / 05311940000021

Departemen : Teknologi Informasi

Dosen Pembimbing 1 / NIP : Henning Titi Ciptaningtyas, S.Kom, M. Kom. /
198407082010122004

Dosen Pembimbing 2 / NIP : Ridho Rahman Hariadi, S.Kom, M. Sc. /
198702132014041001

Dengan ini menyatakan bahwa Tugas Akhir dengan judul

“CLOUD PROVISIONING MENGGUNAKAN GENETIC ALGORITHM DAN ARTIFICIAL NEURAL NETWORK” adalah hasil karya sendiri, bersifat orisinal, dan ditulis dengan mengikuti kaidah penulisan ilmiah.

Bilamana di kemudian hari ditemukan ketidaksesuaian dengan pernyataan ini, maka saya bersedia menerima sanksi sesuai dengan ketentuan yang berlaku di Institut Teknologi Sepuluh Nopember.

Surabaya, 25 Desember 2022

Mengetahui,

Nama Bryan Yehuda Mannuel

NRP 05311940000021



(Mahasiswa)

Nama Henning Titi Ciptaningtyas, S.Kom, M. Kom.

NIP 198407082010122004

(Pembimbing 1)

Nama Ridho Rahman Hariadi, S.Kom, M. Sc.

NIP 198702132014041001

(Pembimbing 2)

STATEMENT OF ORIGINALITY

The undersigned below:

Name of Student / NRP : Bryan Yehuda Mannuel / 05311940000021

Department : Information Technology

Advisor 1 / NIP : Henning Titi Ciptaningtyas, S.Kom, M. Kom. /
198407082010122004

Advisor 2 / NIP : Ridho Rahman Hariadi, S.Kom, M. Sc. /
198702132014041001

Hereby declare that the Final Project with the title of


“CLOUD PROVISIONING MENGGUNAKAN GENETIC ALGORITHM DAN ARTIFICIAL NEURAL NETWORK” is the result of my own work, is original, and is written by following the rules of scientific writing.

If in the future there is a discrepancy with this statement, then I am willing to accept sanctions in accordance with the provisions that apply at Institut Teknologi Sepuluh Nopember.

Surabaya, 25 December 2022

Acknowledged,

Name Bryan Yehuda Mannuel
NRP 05311940000021



(Student)

Name Henning Titi Ciptaningtyas, S.Kom, M. Kom.
NIP 198407082010122004

(Advisor 1)

Name Ridho Rahman Hariadi, S.Kom, M. Sc.
NIP 198702132014041001

(Advisor 2)

CLOUD PROVISIONING MENGGUNAKAN GENETIC ALGORITHM DAN ARTIFICIAL NEURAL NETWORK

Nama Mahasiswa / NRP : Bryan Yehuda Mannuel / 05311940000021
Departemen : Teknologi Informasi
Dosen Pembimbing 1 : Henning Titi Ciptaningtyas, S.Kom, M. Kom.
Dosen Pembimbing 2 : Ridho Rahman Hariadi, S.Kom, M. Sc.

Abstrak

Di era modern dimana penggunaan teknologi semakin pesat dan meningkat secara cepat, penggunaan *Cloud Computing* semakin banyak diminati. Oleh karena itu, diperlukan sebuah sistem manajemen sumber daya yang baik bagi sebuah *Cloud Service Provider* agar sistem *Cloud Computing* mereka dapat memanfaatkan kemampuan virtualisasi sumber daya secara maksimal dan meningkatkan tingkat penggunaan sumber daya *Cloud*. Namun, tantangan terbesar dalam membangun sebuah sistem manajemen sumber daya dalam *Cloud Computing* adalah mencari algoritma yang bisa memaksimalkan penggunaan sumber daya *Cloud*.

Untuk bisa mengatasi tantangan tersebut maka diadakanlah penelitian menggunakan algoritma *Genetic Algorithm* yang terinspirasi dari proses seleksi natural dan implementasi *Artificial Neural Network* yang didasarkan pada jaringan saraf biologis yang membentuk otak untuk membangun sebuah sistem penjadwalan tugas dan alokasi mesin virtual (VM) untuk memaksimalkan penggunaan sumber daya *Cloud*.

Penelitian dilakukan dalam dua skenario, dimana untuk skenario pertama akan digunakan *Genetic Algorithm* saja dan untuk skenario kedua akan digunakan *Genetic Algorithm* bersamaan dengan *Artificial Neural Network*. Pada kedua skenario ini, akan dihasilkan peningkatan efisiensi penjadwalan tugas pada tiap iterasi penggunaan *Genetic Algorithm* ataupun tiap iterasi pelatihan *Artificial Neural Network*. Iterasi ini akan dilakukan secara terus menerus hingga tidak didapati peningkatan efisiensi yang signifikan atau kondisi terminasi telah dipenuhi. Kedua skenario tersebut kemudian akan dilakukan perbandingan untuk mencari tahu mana sistem *Cloud Provisioning* yang lebih efisien berdasarkan berbagai macam parameter penilaian.

Hasil penelitian memberikan kesimpulan bahwa implementasi *Genetic Algorithm* saja bisa menghasilkan tingkat *Resource Utilization* sekitar 48% hingga 60% dan implementasi *Genetic Algorithm* bersamaan dengan *Artificial Neural Network* sekitar 38% hingga 59%. Implementasi *Genetic Algorithm* saja berguna untuk menghemat penggunaan energi, memaksimalkan *Resource Utilization*, dan mengurangi *Execution Time*. Sedangkan implementasi *Genetic Algorithm* bersamaan dengan *Artificial Neural Network* bisa digunakan saat memiliki banyak *Task* dengan ketidakseimbangan data yang besar dan ingin cepat melakukan *Cloud Provisioning*.

Kata Kunci: *Artificial Neural Network, Cloud Computing, Genetic Algorithm, Mesin Virtual, Penjadwalan Tugas*

CLOUD PROVISIONING USING GENETIC ALGORITHM AND ARTIFICIAL NEURAL NETWORK

Name of Student / NRP : Bryan Yehuda Mannuel / 05311940000021
Department : Information Technology
Advisor 1 : Henning Titi Ciptaningtyas, S.Kom, M. Kom. /
Advisor 2 : Ridho Rahman Hariadi, S.Kom, M. Sc.

Abstract

In this modern era where the use of technology is growing rapidly and increasing rapidly, the use of Cloud Computing is increasingly in demand. Therefore, a good resource management system is needed for a Cloud Service Provider so that their Cloud Computing system can take full advantage of resource virtualization capabilities and increase the level of resource utilization of Cloud resources. However, the biggest challenge in building a resource management system in Cloud Computing is to find an algorithm that can maximize the use of Cloud resources.

To be able to overcome these challenges, this research was conducted using a Genetic Algorithm inspired by the natural selection process and the implementation of an Artificial Neural Network which is based on a biological neural network that forms the brain to build a task scheduling and virtual machine (VM) allocation system to maximize the use of cloud resources.

The research was conducted using two different scenarios, the first of which just used the Genetic Algorithm and the second of which combined it with an Artificial Neural Network. With each iteration of the Genetic Algorithm or Artificial Neural Network training in these two scenarios, task scheduling effectiveness will rise. Until no appreciable improvement in efficiency is identified or the termination conditions have been satisfied, this iteration will continue. After that, the two scenarios will be contrasted to determine whether Cloud Provisioning solution is more effective based on several evaluation parameter.

The results of the study conclude that the implementation of a Genetic Algorithm alone can produce a Resource Utilization level of around 48% to 60% and the implementation of a Genetic Algorithm together with an Artificial Neural Network is around 38% to 59%. Genetic Algorithm implementation alone is useful for saving energy use, maximizing Resource Utilization, and reducing Execution Time. Meanwhile, the implementation of a Genetic Algorithm together with an Artificial Neural Network can be used when you have many Tasks with large data imbalances and want to quickly do Cloud Provisioning.

Keyword: *Artificial Neural Network, Cloud Computing, Genetic Algorithm, Task Scheduling, Virtual Machine*

KATA PENGANTAR

Puji syukur penulis ucapkan kepada Tuhan Yang Maha Esa atas pimpinan, penyertaan, dan karunia-Nya sehingga penulis dapat menyelesaikan Tugas Akhir yang berjudul:

Cloud Provisioning Menggunakan Genetic Algorithm dan Artificial Neural Network

Pengerjaan tugas akhir ini dilakukan untuk memenuhi salah satu syarat meraih gelar Sarjana di Departemen Teknologi Informasi Fakultas Teknologi Elektro dan Informatika Cerdas Institut Teknologi Sepuluh Nopember. Dengan selesainya tugas akhir ini diharapkan apa yang telah dikerjakan penulis dapat memberikan manfaat bagi perkembangan ilmu pengetahuan terutama di bidang teknologi informasi serta bagi diri penulis selaku peneliti.

Penulis ingin menyampaikan rasa terima kasih kepada banyak pihak yang telah membimbing, menemani dan membantu penulis selama masa pengerjaan tugas akhir maupun masa studi.

1. Tuhan Yesus Kristus atas semua berkat, hikmat, kekuatan, dan penyertaan yang diberikan kepada penulis selama ini.
2. Kedua orangtua penulis, Bapak Setijo Agus dan Ibu Dewi Natasia yang mendukung, memberi semangat dan mendoakan sehingga penulis bisa sampai di titik ini dan menyelesaikan Tugas Akhir dengan baik.
3. Adik penulis, Brendan Timothy Mannuel yang selalu berada di sisi penulis dan memberikan dukungan moral untuk terus maju dan menyelesaikan Tugas Akhir dengan baik.
4. Bapak Ir. Raden Venantius Hari Ginardi, M.Sc. selaku Kepala Program Studi S1 Teknologi Informasi.
5. Ibu Henning Titi Ciptaningtyas, S.Kom, M. Kom. selaku pembimbing 1 dan Bapak Ridho Rahman Hariadi S.Kom, M.Sc selaku pembimbing 2 yang selalu memberikan arahan dan dukungan untuk kelancaran pelaksanaan tugas akhir Penulis.
6. Teman-teman Programmer, Alvin Putra, Evelyn Sierra, dan Daniel Evan yang memberikan banyak masukan dan saran terkait pelaksanaan tugas akhir.
7. Serta pihak-pihak lain yang penulis tidak dapat disebutkan satu per satu, dan tentunya turut andil dalam membantu penulis selama masa studi.

Penulis memohon maaf apabila dalam tugas akhir ini ada ditemukan kekurangan. Penulis mengharapkan kritik dan saran yang bersifat membangun untuk pembelajaran dan evaluasi di kemudian hari. Semoga dengan adanya tugas akhir ini, penulis dapat memberikan kontribusi dan manfaat yang baik.

Surabaya, 25 Desember 2022



Bryan Yehuda Mannuel

DAFTAR ISI

LEMBAR PENGESAHAN.....	iv
APPROVAL SHEET	vi
PERNYATAAN ORISINALITAS.....	viii
STATEMENT OF ORIGINALITY.....	ix
Abstrak.....	x
Abstract.....	xi
KATA PENGANTAR.....	xii
DAFTAR ISI.....	xiii
DAFTAR GAMBAR.....	xvii
DAFTAR TABEL.....	xviii
DAFTAR KODE SUMBER	xix
BAB I PENDAHULUAN.....	1
1.1 Latar Belakang	1
1.2 Rumusan Masalah.....	2
1.3 Batasan Masalah	2
1.4 Tujuan	2
1.5 Manfaat	2
1.6 Metodologi.....	2
1.6.1 Penyusunan Proposal Tugas Akhir	3
1.6.2 Studi Literatur	3
1.6.3 Analisis dan Desain Perangkat Lunak	3
1.6.4 Implementasi Perangkat Lunak.....	4
1.6.5 Pengujian dan Evaluasi.....	4
1.6.6 Penyusunan Buku Tugas Akhir	4
1.7 Sistematika Penulisan	4
BAB II TINJAUAN PUSTAKA.....	6
2.1 Tinjauan Penelitian Sebelumnya	6
2.2 Dasar Teori	8
2.2.1 Cloud Computing	9
2.2.2 Cloud Provisioning.....	11
2.2.3 Teknologi Virtualisasi	11
2.2.4 Knapsack Problem	12
2.2.5 Genetic Algorithm	13

2.2.6	Artificial Neural Network.....	15
2.2.7	CloudSim.....	19
2.2.8	Library CloudSim	20
2.2.9	Library Encog.....	22
2.2.10	Definisi Parameter Penggunaan Sumber Daya Cloud	22
2.2.11	Tingkatan Penggunaan Sumber Daya.....	24
2.2.12	Dataset The San Diego Supercomputer Center (SDSC) Blue Horizon Log.....	24
2.2.13	Eclipse IDE.....	25
BAB III METODOLOGI		26
3.1	Tahapan Metodologi Penelitian	26
3.2	Deskripsi Metodologi Penelitian	26
3.2.1	Identifikasi Permasalahan	26
3.2.2	Studi Literatur	27
3.2.3	Analisis dan Desain Perangkat Lunak	27
3.2.4	Implementasi Perangkat Lunak.....	32
3.2.5	Pengujian dan Evaluasi.....	33
3.2.6	Penyusunan Buku Tugas Akhir	34
3.3	Jadwal Pengerjaan Tugas Akhir.....	34
BAB IV IMPLEMENTASI		35
4.1	Pengaturan Konfigurasi Simulasi Cloud Pada CloudSim.....	35
4.1.1	Pengaturan Pada Task.....	35
4.1.2	Pengaturan Pada Mesin Virtual (VM)	36
4.1.3	Pengaturan Pada Host	36
4.1.4	Pengaturan Pada Data Center	37
4.2	Pengaturan Parameter Penilaian.....	38
4.2.1	Makespan	38
4.2.2	Average Start Time	38
4.2.3	Average Finish Time	39
4.2.4	Average Execution Time.....	39
4.2.5	Total Wait Time	39
4.2.6	Scheduling Length.....	39
4.2.7	Throughput	40
4.2.8	Resource Utilization	40
4.2.9	Energy Consumption	40
4.2.10	Imbalance Degree.....	40

4.3	Pembuatan Dataset Random	41
4.3.1	Randomize Pada Microsoft Excel	41
4.3.2	Membaca Nilai Dari Microsoft Excel	42
4.3.3	Implementasi Pada Simulasi	43
4.4	Pembuatan Dataset SDSC Blue Horizon Logs	43
4.4.1	Preprocessing Pada Dataset	43
4.4.2	Membaca Nilai Dari Microsoft Excel	44
4.4.3	Implementasi Pada Simulasi	44
4.5	Implementasi Skenario Pertama	45
4.5.1	Pengaturan Kromosom Individu	45
4.5.2	Pengambilan Individu Terbaik di Dalam Populasi	45
4.5.3	Perhitungan Fitness Function	46
4.5.4	Mutation	49
4.5.5	Crossover	49
4.5.6	Implementasi Pada Simulasi	50
4.6	Implementasi Skenario Kedua	51
4.6.1	Preprocessing Pada Dataset	51
4.6.2	Membaca Nilai Dari Microsoft Excel	52
4.6.3	Pembuatan Struktur Artificial Neural Network	54
4.6.4	Normalisasi Data	55
4.6.5	Proses Pembelajaran Data	55
4.6.6	Proses Pengujian Data	56
4.6.7	Implementasi Pada Simulasi	57
BAB V UJI COBA DAN ANALISIS		59
5.1	Makespan	59
5.2	Average Start Time	60
5.3	Average Finish Time	61
5.4	Average Execution Time	62
5.5	Total Wait Time	63
5.6	Scheduling Length	64
5.7	Throughput	65
5.8	Resource Utilization	66
5.9	Total Energy Consumption	67
5.10	Imbalance Degree	68
BAB VI KESIMPULAN DAN SARAN		70

6.1	Kesimpulan	70
6.2	Saran.....	70
DAFTAR PUSTAKA		72
LAMPIRAN.....		75
A.	Kode Sumber <i>CloudSimulationGA.Java</i>	75
B.	Kode Sumber <i>CloudSimulationANN.Java</i>	85
C.	Kode Sumber <i>GeneticAlgorithm.Java</i>	96
D.	Kode Sumber <i>Population.Java</i>	108
E.	Kode Sumber <i>Individual.Java</i>	111
F.	Kode Sumber <i>ANNTTest.Java</i>	114
G.	Dataset <i>Cloudlet</i>	117
H.	Output <i>Genetic Algorithm</i>	117
I.	Dataset <i>Artificial Neural Network</i>	117
J.	Dataset <i>Train Artificial Neural Network</i>	117
K.	Dataset <i>Test Artificial Neural Network</i>	117
L.	Model <i>Artificial Neural Network</i>	117
BIODATA PENULIS		118

DAFTAR GAMBAR

Gambar 2.1 Model Layanan Cloud Computing	10
Gambar 2.2 Knapsack Problem	12
Gambar 2.3 <i>Flowchart</i> Cara Kerja <i>Genetic Algorithm</i>	14
Gambar 2.4 <i>Artificial Neural Network</i> dengan 4 Lapisan	16
Gambar 2.5 <i>Flowchart</i> Cara Kerja <i>Artificial Neural Network</i> Bagian 1	17
Gambar 2.6 <i>Flowchart</i> Cara Kerja <i>Artificial Neural Network</i> Bagian 2.....	18
Gambar 2.7 Melbourne CLOUDS Lab Pengembang CloudSIM	19
Gambar 2.8 Logo <i>Encog</i>	22
Gambar 2.9 Contoh Penjadwalan Tugas	23
Gambar 2.10 Logo <i>Eclipse IDE</i>	25
Gambar 3.1 Representasi Kromosom	28
Gambar 3.2 <i>Flowchart</i> Skenario Pertama	29
Gambar 3.3 <i>Activation Function Sigmoid</i> dan <i>ReLu</i>	30
Gambar 3.4 <i>Flowchart</i> Skenario Kedua	31
Gambar 3.5 Skema Implementasi Perangkat Lunak	32
Gambar 3.6 Skema Implementasi Perangkat Lunak Pembagian VM	32
Gambar 4.1 Hasil Pembuatan Nilai Acak.....	41
Gambar 4.2 Hasil Pemindahan Nilai Acak Pada <i>RandomDataset.txt</i>	42
Gambar 4.3 Dataset SDSC Dari Versi <i>clean-4.2</i>	43
Gambar 4.4 Hasil Bersih Dari <i>Preprocessing</i> Dataset	44
Gambar 4.5 <i>SDSCDataset.txt</i>	44
Gambar 4.6 Dataset Awal Hasil Keluaran <i>Genetic Algorithm</i>	52
Gambar 4.7 Dataset Akhir Setelah <i>Preprocessing</i>	52
Gambar 4.8 Hasil Akhir Setelah Pembagian Dataset	54
Gambar 5.1 Grafik Perbandingan <i>Makespan</i> Pada Dataset Acak	59
Gambar 5.2 Grafik Perbandingan <i>Makespan</i> Pada Dataset SDSC.....	59
Gambar 5.3 Grafik Perbandingan <i>Average Start Time</i> Pada Dataset Acak.....	60
Gambar 5.4 Grafik Perbandingan <i>Average Start Time</i> Pada Dataset SDSC	60
Gambar 5.5 Grafik Perbandingan <i>Average Finish Time</i> Pada Dataset Acak	61
Gambar 5.6 Grafik Perbandingan <i>Average Finish Time</i> Pada Dataset SDSC.....	62
Gambar 5.7 Grafik Perbandingan <i>Average Execution Time</i> Pada Dataset Acak	62
Gambar 5.8 Grafik Perbandingan <i>Average Execution Time</i> Pada Dataset SDSC.....	63
Gambar 5.9 Grafik Perbandingan <i>Total Wait Time</i> Pada Dataset Acak.....	63
Gambar 5.10 Grafik Perbandingan <i>Total Wait Time</i> Pada Dataset SDSC	64
Gambar 5.11 Grafik Perbandingan <i>Scheduling Length</i> Pada Dataset Acak.....	64
Gambar 5.12 Grafik Perbandingan <i>Scheduling Length</i> Pada Dataset SDSC	65
Gambar 5.13 Grafik Perbandingan <i>Throughput</i> Pada Dataset Acak.....	65
Gambar 5.14 Grafik Perbandingan <i>Throughput</i> Pada Dataset SDSC	66
Gambar 5.15 Grafik Perbandingan <i>Resource Utilization</i> Pada Dataset Acak.....	66
Gambar 5.16 Grafik Perbandingan <i>Resource Utilization</i> Pada Dataset SDSC	67
Gambar 5.17 Grafik Perbandingan <i>Total Energy Consumption</i> Pada Dataset Acak	67
Gambar 5.18 Grafik Perbandingan <i>Total Energy Consumption</i> Pada Dataset SDSC.....	68
Gambar 5.19 Grafik Perbandingan <i>Imbalance Degree</i> Pada Dataset Acak	68
Gambar 5.20 Grafik Perbandingan <i>Imbalance Degree</i> Pada Dataset SDSC	69

DAFTAR TABEL

Tabel 2.1 Penelitian Sebelumnya	6
Tabel 2.2 <i>Namespace CloudSim</i>	20
Tabel 2.3 Parameter Tingkat Penggunaan Sumber Daya <i>Cloud</i>	23
Tabel 3.1 Dataset yang Digunakan	27
Tabel 3.2 Spesifikasi <i>Genetic Algorithm</i>	27
Tabel 3.3 <i>Fitness Function</i>	28
Tabel 3.4 Spesifikasi <i>Artificial Neural Network</i>	30
Tabel 3.5 Spesifikasi <i>Node Artificial Neural Network</i>	30
Tabel 3.6 Spesifikasi <i>Virtual Machine</i>	33
Tabel 3.7 Spesifikasi <i>Host</i>	33
Tabel 3.8 Spesifikasi <i>Data Center</i>	33
Tabel 3.9 Parameter yang Digunakan	34
Tabel 3.10 Jadwal Pengerjaan	34
Tabel 4.1 Pembagian Dataset	53
Tabel 4.2 Pembagian Dataset Panjang <i>Task</i>	57

DAFTAR KODE SUMBER

Kode Sumber 4.1 Metode <i>CreateCloudlet</i>	35
Kode Sumber 4.2 Metode <i>CreateVM</i>	36
Kode Sumber 4.3 Metode <i>createDataCenter</i> Bagian Pengaturan <i>Host</i>	37
Kode Sumber 4.4 Metode <i>createDataCenter</i> Bagian Pengaturan <i>Data Center</i>	38
Kode Sumber 4.5 <i>Makespan</i>	38
Kode Sumber 4.6 <i>Average Start Time</i>	38
Kode Sumber 4.7 <i>Average Finish Time</i>	39
Kode Sumber 4.8 <i>Average Execution Time</i>	39
Kode Sumber 4.9 <i>Total Wait Time</i>	39
Kode Sumber 4.10 <i>Scheduling Length</i>	39
Kode Sumber 4.11 <i>Throughput</i>	40
Kode Sumber 4.12 <i>Resource Utilization</i>	40
Kode Sumber 4.13 <i>Energy Consumption</i>	40
Kode Sumber 4.14 <i>Imbalance Degree</i>	41
Kode Sumber 4.15 <i>Randbetween</i>	41
Kode Sumber 4.16 <i>getSeedValue</i>	42
Kode Sumber 4.17 <i>Individual</i>	45
Kode Sumber 4.18 <i>getFittest</i>	46
Kode Sumber 4.19 Perhitungan <i>Fitness Function</i> Bagian Total Waktu Eksekusi	46
Kode Sumber 4.20 <i>Pseudocode</i> Distribusi <i>Poisson</i>	47
Kode Sumber 4.21 Distribusi <i>Poisson</i> Dalam Bahasa <i>Java</i>	47
Kode Sumber 4.22 Distribusi <i>Poisson Redundant</i> Dalam Bahasa <i>Java</i>	48
Kode Sumber 4.23 Bagian Terakhir <i>Fitness Function</i>	48
Kode Sumber 4.24 Potongan Kode <i>Mutation</i>	49
Kode Sumber 4.25 Potongan Kode <i>Crossover</i>	50
Kode Sumber 4.26 Potongan Kode Simulasi <i>Genetic Algorithm</i>	51
Kode Sumber 4.27 Metode Membaca Dari File Txt	54
Kode Sumber 4.28 Pembuatan Struktur <i>Artificial Neural Network</i>	54
Kode Sumber 4.29 Normalisasi Data	55
Kode Sumber 4.30 Proses Pembelajaran <i>Artificial Neural Network</i>	56
Kode Sumber 4.31 Proses Pengujian <i>Artificial Neural Network</i>	56
Kode Sumber 4.32 Potongan Kode Simulasi <i>Artificial Neural Network</i>	58

BAB I

PENDAHULUAN

1.1 Latar Belakang

Di era modern dimana penggunaan teknologi semakin pesat dan meningkat secara cepat, penggunaan *Cloud Computing* semakin banyak diminati (Ray, 2017). *Cloud Computing* adalah ketersediaan sumber daya sistem komputer sesuai permintaan, seperti penyimpanan data dan daya komputasi, tanpa pengelolaan langsung oleh pengguna (Ahmadreza Montazerolghaem, 2020). Namun, penelitian terbaru menyatakan bahwa tingkat penggunaan sumber daya *Cloud* di banyak pusat data masih terbilang cukup rendah. Hal ini diakibatkan karena masih banyak *Cloud Service Provider* yang tidak menggunakan kemampuan virtualisasi yang dimiliki oleh *Cloud Computing* secara maksimal sehingga berakibat kepada pembuangan energi dan tenaga secara sia-sia (Michael Pawlish A. S., 2012). Oleh karena itu, diperlukan sebuah sistem manajemen sumber daya yang baik bagi sebuah *Cloud Service Provider* agar sistem *Cloud Computing* mereka dapat memanfaatkan kemampuan virtualisasi sumber daya secara maksimal dan meningkatkan tingkat penggunaan sumber daya *Cloud*.

Cloud provisioning adalah fitur utama dari sistem *Cloud Computing*, yang berkaitan dengan cara pelanggan mendapatkan sumber daya *Cloud* dari *Cloud Service Provider* (Montgomery, 2020). Penjadwalan tugas dan alokasi mesin virtual (VM) memainkan peran penting dalam *Cloud provisioning*. Hal ini dikarenakan sistem *Cloud Computing* bergantung pada teknologi virtualisasi yang memungkinkan sumber daya dari satu sumber daya *Cloud* fisik untuk dibagi menjadi beberapa lingkungan terisolasi yang berjalan di mesin virtual (VM) (Farouk A. Emara, 2021).

Tantangan terbesar dalam membangun sebuah sistem penjadwalan tugas dan alokasi mesin virtual (VM) dalam *Cloud Computing* adalah mencari algoritma yang bisa memaksimalkan penggunaan sumber daya *Cloud*. Tantangan ini biasa disebut sebagai “*Knapsack Problem*” dimana “Diberikan sekumpulan benda, masing-masing dengan bobot dan nilai tertentu, maka tentukan jumlah setiap benda untuk dimasukkan kedalam koleksi sehingga bobot totalnya kurang dari atau sama dengan batas yang diberikan dan nilai totalnya sebesar mungkin. (G. B. Mathews, 1896)”. Tantangan ini sering muncul dalam pengalokasian sumber daya di mana pengambil keputusan harus memilih dari serangkaian tugas yang tidak dapat dibagi di bawah anggaran tetap atau batasan waktu (Dantzig, 2007).

Untuk bisa mengatasi hal tersebut maka diadakanlah penelitian menggunakan algoritma *Genetic Algorithm* yang terinspirasi dari proses seleksi natural dan implementasi *Artificial Neural Network* yang didasarkan pada jaringan saraf biologis yang membentuk otak untuk membangun sebuah sistem penjadwalan tugas dan alokasi mesin virtual (VM) untuk memaksimalkan penggunaan sumber daya *Cloud*. *Genetic Algorithm* digunakan untuk menghasilkan solusi berkualitas tinggi untuk optimasi penggunaan sumber daya *Cloud* dengan mengandalkan operator yang terinspirasi secara biologis seperti mutasi, penyilangan dan seleksi (Mitchell, 1996). Ditambahkan dengan implementasi *Artificial Neural Network* untuk mempelajari, memproses, dan memprediksi hasil dari solusi optimasi. (Kalita, 2022).

1.2 Rumusan Masalah

Rumusan masalah yang diangkat dalam tugas akhir ini adalah sebagai berikut:

1. Bagaimana cara mengimplementasikan *Genetic Algorithm* untuk menghasilkan solusi berkualitas tinggi untuk optimasi penggunaan sumber daya *Cloud*?
2. Bagaimana cara mengimplementasikan *Artificial Neural Network* bersamaan dengan *Genetic Algorithm* untuk mempelajari, memproses, dan memperkirakan model dari solusi optimasi penggunaan sumber daya *Cloud*?

1.3 Batasan Masalah

Batasan masalah untuk pengerjaan tugas akhir ini adalah sebagai berikut:

1. Proses penelitian akan berupa simulasi *Cloud Environment* dengan menggunakan *CloudSIM Java*, *Library Encog* untuk pengkodean *Artificial Neural Network* dan *Eclipse IDE*.
2. Dataset yang digunakan untuk penelitian ini akan menggunakan dataset yang dibuat sendiri secara acak dan juga mengambil dari dataset *The San Diego Supercomputer Center (SDSC) Blue Horizon logs* (log, 2003).
3. Proses penelitian akan ditulis menggunakan Bahasa *Java* menggunakan *IDE Eclipse*.
4. Proses penelitian ini tidak akan membahas mengenai biaya penggunaan sistem *Cloud Computing*.

1.4 Tujuan

Tujuan dari pengerjaan tugas akhir ini adalah menyelesaikan “*Knapsack Problem*” yang ditemui pada saat melakukan *Cloud Provisioning* menggunakan *Genetic Algorithm* dan *Artificial Neural Network* untuk membangun sebuah sistem penjadwalan tugas dan alokasi mesin virtual (VM) yang bisa memaksimalkan penggunaan sumber daya *Cloud* sehingga meningkatkan tingkat penggunaan sumber daya *Cloud* dan mengurangi energi dan tenaga yang terbuang sia-sia.

1.5 Manfaat

Manfaat dari pengerjaan tugas akhir ini adalah sebagai berikut:

1. Membangun sebuah sistem penjadwalan tugas dan alokasi mesin virtual (VM).
2. Mencegah penurunan performa sumber daya *Cloud*.
3. Meningkatkan tingkat penggunaan sumber daya *Cloud*.
4. Mengurangi *Execution Time* penggunaan sumber daya *Cloud*.
5. Mengurangi energi dan tenaga yang terbuang sia-sia saat penggunaan penggunaan sumber daya *Cloud*.

1.6 Metodologi

Metodologi pengerjaan yang diterapkan pada Tugas Akhir ini mempunyai tahapan-tahapan sebagai berikut

1.6.1 Penyusunan Proposal Tugas Akhir

Penyusunan tugas akhir ini berisi tentang pendahuluan dari tugas akhir yang akan dilaksanakan dimana terdiri dari latar belakang dimana menjelaskan alasan pengambilan judul tugas akhir, rumusan masalah, batasan masalah, tujuan akhir dari tugas akhir, serta manfaat dari tugas akhir. Pada proposal ini juga terdapat juga tinjauan Pustaka yang digunakan dalam referensi pembuatan tugas akhir.

1.6.2 Studi Literatur

Proposal tugas akhir ini menggunakan beberapa literatur yang sudah pernah dibuat sebelumnya seperti “*Genetic-Based Multi-objective Task Scheduling Algorithm in Cloud Computing Environment*” (Farouk A. Emara, 2021), “*Resource provisioning in scalable cloud using bio-inspired artificial neural network model*” (Pradeep Singh RawatPriti, 2020), “*A Call for Energy Efficiency in Data Centers*” (Michael Pawlish A. S., 2014), dan “*Survey on Task Scheduling Methods in Cloud RPS System*” (Henning Titi Ciptaningtyas, 2022).

1.6.3 Analisis dan Desain Perangkat Lunak

Langkah-langkah dari analisis dan desain perangkat lunak yang akan dibuat adalah melakukan analisa dan *preprocessing* pada dataset yang akan digunakan. Hasil dari *preprocessing* pada dataset ini akan menghasilkan dataset yang bersih sehingga siap untuk dilakukan proses selanjutnya. Pada penelitian ini akan dijalankan dua skenario dimana untuk skenario pertama akan dilakukan penjadwalan tugas dan alokasi mesin virtual (VM) menggunakan *Genetic Algorithm* saja dan untuk skenario kedua akan dilakukan dilakukan penjadwalan tugas dan alokasi mesin virtual (VM) menggunakan *Genetic Algorithm* bersamaan dengan *Artificial Neural Network*. Kedua skenario ini nantinya akan dilakukan perbandingan untuk ditemukan kombinasi algoritma mana yang lebih efisien dalam melakukan penjadwalan tugas dan alokasi mesin virtual (VM).

Untuk skenario pertama akan dilakukan inisialisasi populasi awal dari jadwal tugas (*Schedule*) menggunakan *Genetic Algorithm*. Jadwal (*Schedule*) ini akan berisi ID dari tiap mesin virtual (VM) yang dianggap sesuai oleh *Genetic Algorithm* berdasarkan *Fitness Function* sebagai tempat *Task* dijalankan. Jadwal-jadwal ini kemudian akan dilakukan seleksi, mutasi, dan penyilangan menggunakan *Genetic Algorithm* kembali untuk menghasilkan generasi baru yang lebih efisien dari generasi sebelumnya berdasarkan *Fitness Function*.

Untuk skenario kedua, dataset dari Jadwal (*Schedule*) yang telah dihasilkan oleh *Genetic Algorithm* akan dibagi menjadi 2, yaitu dataset pembelajaran dan dataset pengujian. dataset pembelajaran akan dilakukan analisa dan dipelajari terlebih dahulu oleh *Artificial Neural Network* untuk menghasilkan model penjadwalan *Task* kepada mesin virtual (VM) yang sesuai berdasarkan *Fitness Function* milik *Genetic Algorithm*. Model ini nantinya akan dilakukan pengujian menggunakan dataset pengujian untuk mencari tahu apakah penggunaan model ini bisa menghasilkan efisiensi dalam melakukan penjadwalan tugas dan alokasi mesin virtual (VM) yang lebih tinggi daripada penggunaan *Genetic Algorithm* saja.

1.6.4 Implementasi Perangkat Lunak

Implementasi dari perangkat lunak akan menggunakan *CloudSIM*. Sebuah kerangka kerja *Open Source*, yang digunakan untuk mensimulasikan layanan *Cloud Computing* dan menggunakan *Library Encog* untuk pengkodean *Artificial Neural Network*.

1.6.5 Pengujian dan Evaluasi

Pengujian dan evaluasi akan dilaksanakan dengan uji coba menggunakan simulasi *Cloud Environment* yang dijalankan pada *CloudSIM* untuk menguji efisiensi melakukan *Cloud Provisioning* menggunakan algoritma *Genetic Algorithm* dan *Genetic Algorithm* bersamaan dengan *Artificial Neural Network*.

1.6.6 Penyusunan Buku Tugas Akhir

Pada tahap ini akan dilakukan penyusunan laporan yang menjelaskan dasar teori dan metode yang digunakan dalam tugas akhir ini serta hasil dari implementasi aplikasi perangkat lunak yang telah dibuat.

1.7 Sistematika Penulisan

Buku Tugas Akhir ini merupakan laporan secara lengkap mengenai Tugas Akhir yang telah dikerjakan baik dari sisi teori, rancangan, maupun implementasi sehingga memudahkan bagi pembaca dan juga pihak yang ingin mengembangkan lebih lanjut. Sistematika penulisan buku Tugas Akhir secara garis besar antara lain:

1. Bab I : Pendahuluan

Bab ini berisikan penjelasan mengenai latar belakang, rumusan masalah, Batasan masalah, tujuan, manfaat, metodologi, dan sistematika penulisan dari pembuatan Tugas Akhir.

2. Bab II : Dasar Teori

Bab ini berisi kajian teori dari metode dan algoritma yang digunakan dalam penyusunan Tugas Akhir ini.

3. Bab III : Desain

Bab ini menjelaskan desain algoritma yang akan dibangun berdasarkan dasar teori dan digunakan dalam penyelesaian.

4. Bab IV : Implementasi

Bab ini membahas implementasi dari perancangan yang telah dibuat pada bab sebelumnya. Terdapat juga penjelasan berupa kode program yang digunakan untuk proses implementasi.

5. Bab V : Uji Coba dan Analisis

Bab ini membahas tahapan uji coba, kemudian hasil uji coba dievaluasi dan dianalisis terhadap kinerja dari algoritma yang diimplementasikan.

6. Bab VI : Kesimpulan dan Saran

Bab ini merupakan bab yang menjelaskan kesimpulan dari hasil uji coba yang dilakukan, masalah-masalah yang dialami pada proses dan tertulis saat pengerjaan Tugas Akhir, dan saran untuk pengembangan solusi ke depannya.

BAB II

TINJAUAN PUSTAKA

2.1 Tinjauan Penelitian Sebelumnya

Dalam penelitian ini, akan digunakan beberapa penelitian terdahulu sebagai pedoman dan referensi dalam melaksanakan proses pengerjaan tugas akhir. Informasi yang disampaikan dalam Tabel 2.1 berisi informasi mengenai penelitian sebelumnya, hasil penelitian, dan hubungannya terhadap tugas akhir.

Tabel 2.1 Penelitian Sebelumnya

No	Tahun, Penulis	Pembahasan
1	<i>Genetic-Based Multi-objective Task Scheduling Algorithm in Cloud Computing Environment</i> (Farouk A. Emara, 2021)	
	Farouk A. Emara, Ahmed. A. A. Gad-Elrab, Ahmed Sobhi, K. R. Raslan	<p>Jurnal ini membahas mengenai berbagai macam tantangan dalam membangun sebuah sistem penjadwalan tugas dan alokasi mesin virtual (VM) dalam sebuah sistem <i>Cloud Computing</i>. Hal ini dikarenakan banyaknya heterogenitas dalam sistem <i>Cloud Computing</i> yang mengakibatkan banyak algoritma dan penelitian sebelumnya hanya memperhitungkan satu target capaian saja. Jurnal ini menyarankan penggunaan <i>Genetic Algorithm</i> yang dimodifikasi untuk bisa menyelesaikan lebih dari satu target capaian yaitu memaksimalkan <i>Resource Utilization</i>, <i>Load Balancing</i>, dan <i>Power Management</i> dalam sebuah sistem <i>Cloud Computing</i>.</p> <p>Algoritma yang disarankan ini menggunakan sebuah struktur matriks untuk merepresentasikan kromosom dari GAS (komponen yang dicari optimisasinya dalam penggunaan <i>Genetic Algorithm</i>) yang terdiri dari <i>ID Tasks</i>, <i>ID VM</i>, dan <i>ID Data Center</i>. Hasil penelitian dari jurnal ini menunjukkan bahwa penggunaan <i>Genetic Algorithm</i> yang dimodifikasi akan menghasilkan performa penjadwalan tugas dan alokasi mesin virtual (VM) yang lebih baik dari beberapa algoritma lainnya (<i>ETVMC</i>, <i>TSACS</i>, dan <i>ACO</i>) dinilai dari <i>Makespan</i>, <i>Scheduling Length</i>, <i>Throughput</i>, <i>Resource Utilization</i>, <i>Energy Consumption</i>, dan <i>Imbalance Degree</i>.</p> <p>Jurnal ini dijadikan referensi untuk penelitian ini dikarenakan hasil dari penelitiannya membuktikan bahwa penggunaan <i>Genetic Algorithm</i> yang dimodifikasi bisa memberikan hasil yang lebih baik daripada algoritma penjadwalan tugas dan alokasi mesin virtual (VM) lainnya. Penulis mengambil referensi ini dan ingin membandingkan apakah jika <i>Genetic Algorithm</i> yang dibantu dengan <i>Artificial Neural Network</i> bisa memberikan hasil yang lebih baik lagi daripada dengan <i>Genetic Algorithm</i> saja jika dinilai dari <i>Makespan</i>, <i>Scheduling Length</i>, <i>Throughput</i>, <i>Resource Utilization</i>, <i>Energy Consumption</i>, dan <i>Imbalance Degree</i>.</p>

No	Tahun, Penulis	Pembahasan
		Terakhir, penulis juga mengambil referensi penggunaan <i>ID Tasks</i> , <i>ID VM</i> , dan <i>ID Data Center</i> sebagai representasi kromosom dari penggunaan <i>Genetic Algorithm</i> .
2	<i>Resource Provisioning in Scalable Cloud Using Bio-inspired Artificial Neural Network Model</i> (Pradeep Singh Rawat, 2020)	
	Pradeep Singh Rawat, Priti Dimri, Punit Gupta, G.P. Saroha	<p>Jurnal ini membahas mengenai berbagai macam parameter yang bisa digunakan untuk mengukur efisiensi sebuah algoritma penjadwalan tugas dan alokasi mesin virtual (VM) dalam sebuah sistem <i>Cloud Computing</i>. Parameter tersebut contohnya adalah <i>Average Start Time</i>, <i>Average Finish Time</i>, <i>Average Execution Time</i>, dan <i>Wait Time</i>. Parameter ini kemudian digunakan untuk membandingkan beberapa algoritma yaitu <i>BB-BC</i>, <i>GA-Cost</i>, <i>GA-Exe</i>, dan <i>GA-ANN</i> (<i>Genetic Algorithm – Artificial Neural Network</i>) sebagai algoritma utama yang dihipotesiskan penelitian ini menjadi yang terbaik diantara algoritma lainnya.</p> <p>Penelitian ini pun dilakukan menggunakan beberapa variabel kontrol seperti jumlah <i>Task</i>, jumlah <i>VM</i>, jumlah <i>Data Center</i>, <i>Bandwith</i>, <i>CPU</i>, <i>Ram</i>, dan lain sebagainya. Hal ini berguna agar nantinya perbandingan antar algoritma bisa dilaksanakan secara objektif. Penelitian ini juga menggunakan dua sumber data yaitu menggunakan data buatan sendiri dan juga dataset <i>The San Diego Supercomputer Center (SDSC) Blue Horizon logs</i>. Pada akhirnya ditemukan bahwa benar algoritma <i>GA-ANN</i> bisa menghasilkan sistem penjadwalan tugas dan alokasi mesin virtual (VM) yang paling efisien dengan pengurangan 82.63% pada tingkat kesalahan yang terjadi, peningkatan 26.81% pada jumlah <i>Task</i> yang berhasil diselesaikan secara sukses, 10.66% pada <i>Execution Time</i>, dan 69.94% pada <i>Wait Time</i>.</p> <p>Jurnal ini dijadikan referensi untuk penelitian ini dikarenakan hasil dari penelitiannya membuktikan bahwa penggunaan <i>GA-ANN</i> bisa memberikan hasil yang lebih baik daripada algoritma penjadwalan tugas dan alokasi mesin virtual (VM) lainnya yang dibahas. Penulis mengambil referensi ini dan ingin membandingkan apakah jika <i>Genetic Algorithm</i> yang dibantu dengan <i>Artificial Neural Network</i> bisa memberikan hasil yang lebih baik lagi daripada dengan <i>Genetic Algorithm</i> saja jika dinilai dari <i>Average Start Time</i>, <i>Average Finish Time</i>, <i>Average Execution Time</i>, dan <i>Wait Time</i>. Penulis juga mengambil referensi penggunaan dua sumber data yaitu menggunakan data buatan sendiri dan juga dataset <i>The San Diego Supercomputer Center (SDSC) Blue Horizon logs</i>. Terakhir, penulis juga ingin membuat lingkungan percobaan yang sama dengan menggunakan variabel kontrol yang sudah didefinisikan dalam penelitian ini seperti jumlah <i>Task</i>, jumlah <i>VM</i>, jumlah <i>Data Center</i>, <i>Bandwith</i>, <i>CPU</i>, <i>Ram</i>, dan lain sebagainya.</p>

No	Tahun, Penulis	Pembahasan
3	<i>A Call for Energy Efficiency in Data Centers</i> (Michael Pawlish A. S., 2014)	
	Michael Pawlish, Aparna S. Varde, Stefan A. Robila, Anand Ranganathan	<p><i>Conference Paper</i> ini berusaha mencari tahu apakah sistem <i>Cloud Computing</i> yang ada saat ini sudah memiliki tingkat efisiensi penggunaan energi yang sudah tinggi atau belum. Beberapa parameter yang digunakan untuk menilai tingkat efisiensi ini adalah penggunaan energi dan efektivitas. Kedua parameter ini nantinya akan menghasilkan <i>Utilization Rate</i> yang menggambarkan persentase efisiensi penggunaan sistem <i>Cloud Computing</i>. Penelitian ini dilakukan menggunakan data nyata dengan pengaturan universitas.</p> <p>Didapati bahwa <i>Utilization Rate</i> sistem <i>Cloud Computing</i> yang berjalan selama 6 bulan hanya menghasilkan rata-rata 30% hingga 42% saja. Sebuah nilai yang cukup rendah jika dibandingkan dengan saran <i>Utilization Rate</i> yang efisien yaitu di sekitar 70% hingga 80%. <i>Conference Paper</i> ini menyatakan bahwa rendahnya <i>Utilization Rate</i> sistem <i>Cloud Computing</i> ini diakibatkan karena <i>Cloud Service Provider</i> tidak menggunakan kemampuan virtualisasi yang dimiliki oleh <i>Cloud Computing</i> secara maksimal sehingga berakibat kepada pembuangan energi dan tenaga secara sia-sia.</p> <p><i>Conference Paper</i> ini memanggil para <i>Cloud Service Provider</i> dan juga penulis secara tidak langsung agar segera bisa menyelesaikan permasalahan ini. Apabila permasalahan ini tidak segera diselesaikan, akan terjadi sangat banyak pembuangan energi dan tenaga secara sia-sia. <i>Conference Paper</i> ini menjadi tujuan utama penulis ingin membuat sebuah sistem penjadwalan tugas dan alokasi mesin virtual (VM) agar bisa meningkatkan <i>Utilization Rate</i> sistem <i>Cloud Computing</i>.</p>
4	<i>Survey on Task Scheduling Methods</i> (Henning Titi Ciptaningtyas, 2022)	
	Henning Titi Ciptaningtyas, Ary Mazharuddin Shiddiqi, Diana Purwitasari	<p>Jurnal ini menyediakan berbagai macam parameter yang bisa digunakan untuk mengukur efisiensi sistem penjadwalan tugas dan alokasi mesin virtual (VM). Jurnal ini juga menyediakan berbagai macam definisi mengenai parameter-parameter yang akan digunakan dalam penelitian ini seperti <i>Makespan</i>, <i>Throughput</i>, <i>Resource Utilization</i>. Jurnal ini juga menyediakan rumus dan cara menghitung berbagai parameter diatas. Karena semua alasan ini, penulis memilih menggunakan jurnal ini sebagai referensi dalam penelitian ini.</p>

2.2 Dasar Teori

Berikut ini merupakan dasar teori yang akan digunakan oleh penulis dalam penelitian tugas akhir ini. Bagian ini penting untuk memahami apa saja pengertian dasar yang akan digunakan oleh peneliti dalam menjalankan penelitian ini

2.2.1 Cloud Computing

Cloud Computing adalah ketersediaan sumber daya sistem komputer sesuai permintaan, seperti penyimpanan data dan daya komputasi, tanpa pengelolaan langsung oleh pengguna (Ahmadreza Montazerolghaem, 2020). *Cloud Computing* bergantung pada pembagian sumber daya untuk mencapai koherensi dan biasanya menggunakan model "bayar sesuai penggunaan" yang dapat membantu mengurangi biaya modal, tetapi juga dapat menyebabkan biaya operasional yang tidak terduga bagi pengguna yang tidak sadar (Wray, 2014).

Cloud Computing memungkinkan perusahaan untuk menghindari atau meminimalkan biaya infrastruktur di muka. *Cloud Computing* juga memungkinkan perusahaan untuk menjalankan aplikasi mereka lebih cepat, dengan peningkatan pengelolaan dan pemeliharaan yang lebih sedikit, dan memungkinkan perusahaan untuk lebih cepat menyesuaikan sumber daya untuk memenuhi permintaan yang berfluktuasi dan tidak dapat diprediksi (AWS, 2013).

Tujuan *Cloud Computing* adalah untuk memungkinkan pengguna mengambil manfaat dari semua teknologi komputasi terbaru, tanpa perlu pengetahuan mendalam tentang teknologi komputasi tersebut. *Cloud Computing* juga bertujuan untuk memotong biaya dan membantu pengguna fokus pada bisnis inti mereka daripada terhalang oleh hambatan teknologi (Mohammad Hamdaqa, 2012).

Teknologi pendukung utama untuk *Cloud Computing* adalah virtualisasi. Perangkat lunak virtualisasi memisahkan perangkat komputasi fisik menjadi satu atau lebih perangkat "virtual", yang masing-masing dapat dengan mudah digunakan dan dikelola untuk melakukan tugas komputasi. Dengan demikian, sumber daya komputasi yang tidak terpakai dapat dialokasikan dan digunakan secara lebih efisien. Virtualisasi memberikan kelincahan yang diperlukan untuk mempercepat operasi komputasi dan mengurangi biaya dengan meningkatkan pemanfaatan infrastruktur (Mohammad Hamdaqa, 2012).

"Lima Karakteristik Penting" tentang *Cloud Computing* dari Definisi National Institute of Standards and Technology adalah (Technology, 2011):

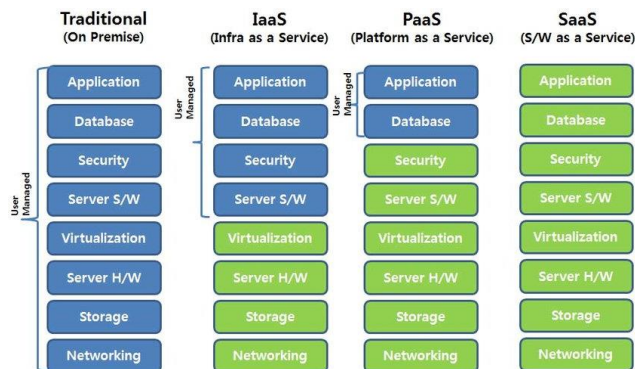
1. *On-demand self-service* dimana konsumen dapat secara langsung menyediakan kemampuan komputasi sesuai kebutuhan secara otomatis tanpa memerlukan interaksi manusia.
2. *Broad network access* dimana kemampuan komputasi tersedia melalui jaringan dan dapat diakses melalui berbagai macam mekanisme (komputer, seluler, server, dll)
3. *Resource Pooling* dimana seluruh sumber daya komputasi milik sebuah *Cloud Service Provider* dikumpulkan untuk melayani banyak konsumen secara sekaligus, dengan sumber daya fisik dan virtual yang berbeda dapat dipindahkan secara dinamis sesuai dengan permintaan konsumen.
4. *Rapid Elasticity* dimana sumber daya komputasi dapat secara elastis disediakan dan dilepaskan sesuai permintaan konsumen
5. *Measured Service* dimana Sistem *Cloud Computing* secara otomatis mengontrol dan mengoptimalkan penggunaan sumber daya dengan memanfaatkan kemampuan pengukuran yang sesuai dengan jenis layanan yang digunakan.

Ada empat model *Cloud Computing* dari Definisi National Institute of Standards and Technology yaitu (Technology, 2011):

1. *Private Cloud* - Infrastruktur *Cloud* yang disediakan secara khusus untuk digunakan oleh satu organisasi yang terdiri dari beberapa unit bisnis. *Private Cloud* dimiliki, dikelola dan dioperasikan oleh satu organisasi, pihak ketiga, atau kombinasi keduanya, dan dapat berada pada satu tempat yang sama ataupun berbeda.
2. *Community Cloud* - Infrastruktur *Cloud* yang disediakan secara khusus untuk digunakan oleh komunitas yang spesifik dari organisasi-organisasi yang memiliki kepentingan bersama. *Community Cloud* dimiliki, dikelola dan dioperasikan oleh satu atau lebih organisasi dalam komunitas tersebut, pihak ketiga, atau kombinasi keduanya, dan dapat berada pada satu tempat yang sama ataupun berbeda.
3. *Public Cloud* - Infrastruktur *Cloud* yang disediakan secara terbuka untuk digunakan oleh masyarakat umum. *Public Cloud* dimiliki, dikelola dan dioperasikan oleh perusahaan, akademis, atau organisasi pemerintah, atau kombinasi dari semuanya. *Public Cloud* berada pada tempat yang ditentukan *Cloud Service Provider*.
4. *Hybrid Cloud* - Infrastruktur *Cloud* yang terdiri dari dua atau lebih infrastruktur *Cloud* yang berbeda (*private*, *community* atau *public*) yang tetap unik, namun terikat pada standar atau paten teknologi yang memungkinkan portabilitas pada data dan aplikasi.

Terdapat tiga macam model *Cloud Computing* utama *Computing* dari Definisi National Institute of Standards and Technology yaitu (Technology, 2011):

1. Infrastructure-as-a-Service (IaaS) dimana perangkat keras komputer disediakan dan dikelola oleh *Cloud Service Provider*. Model ini pada dasarnya ditawarkan kepada mereka yang membutuhkan infrastruktur komputer tetapi tidak ingin berurusan dengan kesulitan mengelolanya.
2. Software-as-a-Service (SaaS) dimana perangkat lunak dilisensikan kepada pelanggan yang membayar. Aplikasi perangkat lunak ini akan *di-host* di Internet, dan pelanggan yang membayar dapat terhubung ke situs *web* tersebut untuk menggunakan perangkat lunak tersebut.
3. Platform-as-a-Service (PaaS) dimana produk model ini adalah platform pengembangan aplikasi berbasis *Cloud*. Model ini memungkinkan pelanggan mengembangkan dan menjalankan aplikasi mereka sendiri tanpa membangun infrastruktur dan menghabiskan uang untuk komponen dan alat yang biasanya mereka perlukan untuk membangun aplikasi mereka.



Gambar 2.1 Model Layanan *Cloud Computing* (Kwangwoog Jung, 2017)

2.2.2 Cloud Provisioning

Cloud provisioning adalah fitur utama dari model *Cloud Computing*, yang berkaitan dengan cara pelanggan mendapatkan sumber daya *Cloud* dari *Cloud Service Provider* (Montgomery, 2020). Penjadwalan tugas dan alokasi mesin virtual (VM) memainkan peran penting dalam *Cloud provisioning*.

Cloud Provisioning memiliki banyak sekali manfaat bagi mereka yang menggunakannya. Manfaat pertama adalah skalabilitas. Dalam model penyediaan kemampuan komputasi tradisional, organisasi memerlukan investasi yang besar dalam menyediakan infrastruktur lokalnya. Hal ini memerlukan persiapan dan perkiraan kebutuhan infrastruktur yang ekstensif, karena infrastruktur lokal sering kali disiapkan untuk bertahan selama beberapa tahun. Namun, menggunakan *Cloud Provisioning*, organisasi dapat dengan mudah meningkatkan dan menurunkan sumber daya *Cloud* mereka.

Pengembang juga dapat memanfaatkan kecepatan *Cloud Provisioning*. Misalnya, pengembang aplikasi dapat dengan cepat menjalankan serangkaian beban kerja sesuai permintaan, dan dengan demikian menghilangkan kebutuhan akan seorang *administrator* yang menyediakan dan mengelola sumber daya komputasi.

Manfaat lain dari *Cloud Provisioning* adalah potensi penghematan biaya. Sementara penyediaan kemampuan komputasi tradisional dapat menuntut investasi awal yang besar dari suatu organisasi, banyak *Cloud Service Provider* memungkinkan pelanggan membayar hanya apa yang mereka gunakan (Montgomery, 2020).

2.2.3 Teknologi Virtualisasi

Virtualisasi adalah teknik bagaimana seseorang bisa memisahkan layanan dari ketersediaan fisik yang mendasari layanan itu. Virtualisasi adalah proses membuat versi virtual dari sesuatu hal yang fisik seperti perangkat keras komputer. Virtualisasi melibatkan penggunaan perangkat lunak khusus untuk membuat versi sumber daya komputasi menjadi virtual. Dengan bantuan Virtualisasi, beberapa sistem operasi dan aplikasi dapat berjalan pada mesin yang sama dan perangkat keras yang sama pada saat yang sama, meningkatkan pemanfaatan dan fleksibilitas perangkat keras (GeeksForGeeks, 2022).

Dengan kata lain, salah satu teknik utama untuk menghemat biaya, mengurangi perangkat keras, dan menghemat energi yang digunakan oleh *Cloud Service Provider* adalah virtualisasi. Hal ini dikarenakan virtualisasi memungkinkan untuk berbagi satu perangkat fisik di antara banyak pelanggan dan organisasi pada waktu yang sama. Hal ini dilakukan dengan menetapkan sebuah *ID* ke penyimpanan fisik dan memberikan *Pointer* ke sumber daya fisik tersebut sesuai permintaan pengguna. Selain itu, teknologi virtualisasi menyediakan lingkungan virtual tidak hanya untuk menjalankan aplikasi saja tetapi juga untuk penyimpanan, memori, dan jaringan (GeeksForGeeks, 2022).

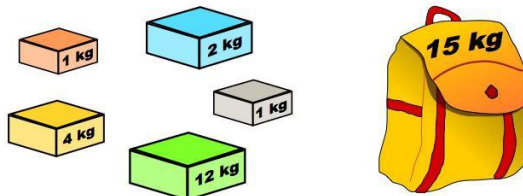
Beberapa manfaat virtualisasi yang sangat penting agar sistem *Cloud Computing* dapat berjalan dengan lancar antara lain (GeeksForGeeks, 2022):

1. Alokasi sumber daya yang lebih fleksibel dan efisien.
2. Meningkatkan produktivitas.
3. Menurunkan biaya infrastruktur teknologi informasi.

4. Akses jarak jauh dan skalabilitas yang cepat.
5. Ketersediaan tinggi dan pemulihan bencana.
6. Sistem pembayaran yang fleksibel
7. Dapat menjalankan banyak sistem operasi dalam satu perangkat fisik.

2.2.4 Knapsack Problem

Tantangan terbesar dalam membangun sebuah sistem penjadwalan tugas dan alokasi mesin virtual (VM) dalam *Cloud Computing* adalah mencari algoritma yang bisa memaksimalkan penggunaan sumber daya *Cloud*. Tantangan ini biasa disebut sebagai “*Knapsack Problem*” dimana “Diberikan sekumpulan benda, masing-masing dengan bobot dan nilai tertentu, maka tentukan jumlah setiap benda untuk dimasukkan kedalam koleksi sehingga bobot totalnya kurang dari atau sama dengan batas yang diberikan dan nilai totalnya sebesar mungkin. (G. B. Mathews, 1896)”.



Gambar 2.2 *Knapsack Problem* (Terh, 2019)

Salah satu contoh dari *Knapsack Problem* adalah dalam penilaian hasil tes di mana peserta tes diberikan pilihan untuk memilih pertanyaan mana yang akan mereka jawab. Untuk jumlah pertanyaan yang sedikit, ini adalah proses yang cukup sederhana. Misalnya, jika ujian berisi 12 pertanyaan yang masing-masing bernilai 10 poin, peserta tes hanya perlu menjawab 10 pertanyaan untuk mencapai skor maksimum 100 poin. Namun, pada tes dengan distribusi nilai poin yang heterogen, akan lebih sulit untuk memperkirakan berapa pertanyaan yang harus dijawab dengan benar. Belum lagi jika jumlah pertanyaan ini diperbesar yang mana hal ini akan semakin menambah kompleksitas perkiraan (Martin Feuerman, 1973).

Knapsack Problem dapat juga penulis dapati saat penulis berusaha melakukan *Cloud Provisioning* karena penulis membutuhkan sebuah sistem yang dapat menjadwalkan tugas dan mengalokasikan mesin virtual (VM) yang tidak dapat dibagi di bawah anggaran tetap dan batasan waktu secara efisien. Variasi dari *Knapsack Problem* yang akan ditemukan pada saat berusaha melakukan *Cloud Provisioning* adalah *Bounded Knapsack Problem* dimana terdapat sejumlah salinan sebanyak X_i untuk setiap barang yang ada. X_i dibatasi hingga sebuah angka integer positif. Berikut adalah penulisan dari *Bounded Knapsack Problem* jika diberikan sebuah kumpulan n benda bernomorkan 1 hingga n , masing-masing dengan nilai V_i dan berat w_i , dengan kapasitas maksimum W , maka Maksimalkan $\sum_{i=1}^n V_i X_i$ dengan memperhatikan $\sum_{i=1}^n w_i X_i \leq W$ dan $X_i \in (0, 1, 2, 3, \dots, C)$.

Hal ini bisa dilihat pada saat penulis ingin melakukan *Cloud Provisioning* dimana sumber daya *Cloud* yang dimiliki oleh sebuah *Cloud Service Provider* pasti lebih dari satu, masing-masing dengan nilai dan biaya mereka sendiri, tetapi tidak mungkin jumlahnya tidak terbatas dan tidak mungkin jumlahnya negatif. Sumber daya *Cloud* ini juga pasti akan dibatasi

oleh kemampuan sewa dari pengguna dimana mereka tidak memiliki keuangan yang tidak terbatas untuk menyewa semua Sumber daya *Cloud* yang dimiliki sebuah *Cloud Service Provider*. Oleh karena itu jika penulis ingin mencari algoritma yang bisa memaksimalkan penggunaan sumber daya *Cloud*, penulis harus berusaha menyelesaikan *Bounded Knapsack Problem* juga.

2.2.5 Genetic Algorithm

Genetic Algorithm adalah sebuah algoritma yang digunakan untuk menghasilkan solusi berkualitas tinggi untuk optimasi penggunaan sumber daya *Cloud* dengan mengandalkan operator yang terinspirasi secara biologis seperti seleksi, mutasi, dan penilangan (Mitchell, 1996). Dalam *Genetic Algorithm*, sebuah populasi dari kandidat solusi (bisa disebut individu, makhluk, organisme, atau fenotipe) untuk sebuah masalah optimasi akan dikembangkan ke arah solusi yang lebih baik. Setiap kandidat solusi memiliki seperangkat sifat (kromosom atau genotipenya) yang dapat dimutasi dan diubah. Sifat ini direpresentasikan dalam biner sebagai string 0 dan 1 (Whitley, 1994).

Proses evolusi ini akan dimulai dari populasi individu yang dihasilkan secara acak, dan merupakan proses berulang, dengan populasi di setiap iterasi yang dihasilkan disebut sebagai generasi. Di setiap generasi, kecocokan setiap individu dalam populasi akan dievaluasi. Kecocokan ini merupakan sebuah nilai dari fungsi tujuan dalam masalah optimasi yang ingin dipecahkan yang disebut sebagai *Fitness Function*. Individu yang lebih cocok akan dipilih secara stokastik dari populasi saat ini, dan genom dari setiap individu akan dimodifikasi (dikombinasikan kembali dan mungkin bermutasi secara acak) untuk membentuk generasi baru. Generasi baru dari kandidat solusi akan digunakan dalam iterasi *Genetic Algorithm* berikutnya. Umumnya, *Genetic Algorithm* akan berakhir ketika jumlah generasi maksimum telah dihasilkan, atau tingkat kecocokan yang memuaskan telah tercapai (Whitley, 1994).

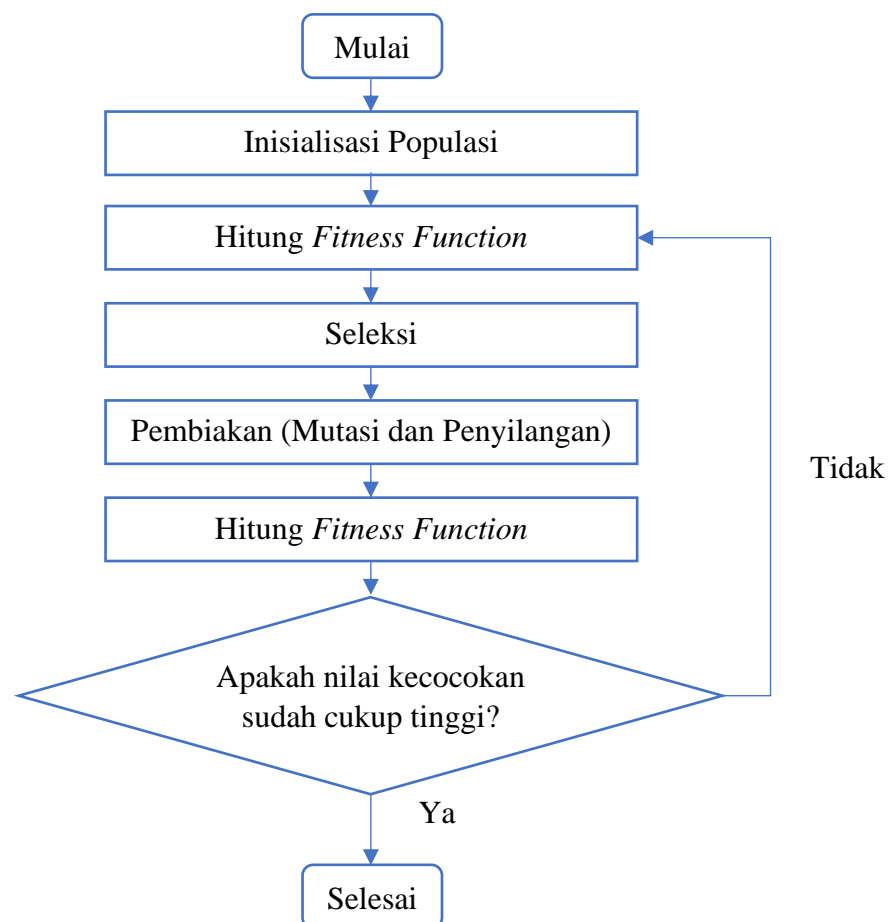
Sebuah *Genetic Algorithm* membutuhkan:

1. Representasi genetik dari kandidat solusi. Representasi ini biasanya dituliskan dalam bentuk *Array of Bits* dengan panjang yang tetap. *Array of Bits* ini akan berisikan bilangan biner 1 dan 0 yang merepresentasikan genetika dari sebuah kandidat solusi (Whitley, 1994).
2. *Fitness Function* untuk menilai kecocokan dari kandidat solusi. Semakin cocok sebuah kandidat solusi dalam masalah optimasi yang ingin dipecahkan maka akan semakin tinggi nilai dari *Fitness Function*-nya (Whitley, 1994).

Proses kerja dari *Genetic Algorithm* terdiri dari beberapa langkah yakni:

1. Inisialisasi dimana populasi awal dari kandidat solusi akan dihasilkan secara acak. Jumlah populasi dari kandidat solusi ini biasanya berjumlah ratusan hingga ribuan pada awalnya namun bisa disesuaikan dengan masalah optimasi yang ingin dipecahkan.
2. Seleksi dimana populasi awal dari kandidat solusi akan dilakukan pemilihan untuk dilakukan pembiakan dan menciptakan generasi baru dari kandidat solusi. Setiap kandidat solusi akan dilakukan seleksi sesuai dengan kecocokan mereka, dinilai dari *Fitness Function*, dimana semakin cocok mereka dengan masalah optimasi yang ingin dipecahkan maka akan semakin tinggi kemungkinan mereka terpilih untuk dilakukan pembiakan.

3. Pembiakan dimana *Genetic Algorithm* akan menghasilkan generasi berikutnya dari kandidat solusi menggunakan kombinasi dari operasi genetika, seperti mutasi dan penyilangan. Dua kandidat solusi “orangtua” (generasi sebelumnya) akan dipilih dan dilakukan pembiakan untuk menghasilkan kandidat solusi “anak” (generasi berikutnya). Pasangan orangtua-orangtua yang baru akan dilakukan pemilihan untuk menghasilkan kombinasi anak yang baru hingga tercapai populasi generasi yang baru.
4. Iterasi yang berarti akan dilakukan proses Seleksi dan Pembiakan generasi yang baru lagi. Dengan melakukan hal ini, kandidat solusi “anak” yang dihasilkan pada setiap generasi baru akan memiliki karakteristik dari orangtua mereka. Berkat dilakukan proses Seleksi, maka dipastikan kandidat solusi “orangtua” yang terpilih akan memiliki nilai kecocokan yang tinggi sesuai dengan *Fitness Function* yang diinginkan. Hal ini akan berakibat meningkatnya nilai kecocokan rata-rata dari setiap generasi baru yang dihasilkan melalui proses Seleksi dan Pembiakan di tiap iterasi.
5. Terminasi dimana ketika sebuah nilai kecocokan dari kandidat solusi dinilai sudah cukup tinggi, maka proses iterasi *Genetic Algorithm* akan dilakukan penghentian. Kandidat solusi yang memiliki nilai kecocokan tinggi tersebut akan dijadikan solusi dari masalah optimasi yang ingin dipecahkan.



Gambar 2.3 Flowchart Cara Kerja *Genetic Algorithm*

Genetic Algorithm dipilih untuk bisa menyelesaikan *Knapsack Problem* yang ditemui pada saat ingin melakukan *Cloud Provisioning* karena beberapa alasan, yaitu:

1. *Genetic Algorithm* sering digunakan untuk menghasilkan solusi berkualitas tinggi untuk optimasi penggunaan sumber daya di mana pengambil keputusan harus memilih dari serangkaian tugas yang tidak dapat dibagi di bawah anggaran tetap atau batasan waktu (Mitchell, 1996).
2. Pengkodean genetik pada *Genetic Algorithm* berbasis biner 1 dan 0 yang sangat cocok digunakan untuk merepresentasikan jumlah barang pada *Knapsack Problem* yang juga dikodekan dengan basis biner 1 dan 0.
3. *Genetic Algorithm* memiliki prosedur penilaian kecocokan kandidat solusi menggunakan *Fitness Function* yang bisa dimodifikasi sesuai dengan keperluan secara fleksibel dan bisa disesuaikan dengan permintaan pengguna saat melakukan *Cloud Provisioning*.
4. Banyak iterasi dari *Genetic Algorithm* bisa disesuaikan dengan kebutuhan untuk mencapai nilai kecocokan yang diinginkan untuk optimasi penggunaan sumber daya *Cloud*.
5. Proses pembiakan dari *Genetic Algorithm* bisa menggunakan berbagai macam operator biologis yang bisa disesuaikan dengan data dan kebutuhan (mutasi, penyilangan, kolonisasi, kepunahan, dll).

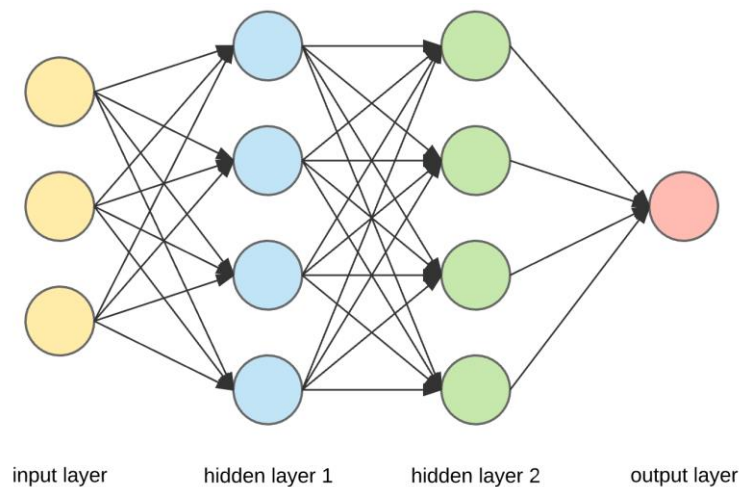
2.2.6 Artificial Neural Network

Neural Network merupakan kumpulan unit atau simpul yang saling terhubung dan didasarkan pada model neuron-neuron yang ada di dalam otak biologis. Kumpulan unit atau simpul ini disebut sebagai neuron buatan. Neuron-neuron buatan ini dapat menerima, memproses dan meneruskan sinyal pada neuron buatan lain yang terhubung dengannya. "Sinyal" ini adalah bilangan real, dan *output* dari setiap neuron buatan dihitung menggunakan berbagai fungsi non-linier dari jumlah *input* yang diterimanya (Hardesty, 2017).

Sambungan antar setiap neuron buatan disebut juga sebagai *edge*. Setiap neuron dan *edge* memiliki bobot yang dapat menyesuaikan diri pada saat proses pembelajaran berlangsung. Bobot ini dapat menambah atau mengurangi kekuatan sinyal pada setiap sambungan antar neuron buatan. Neuron-neuron buatan ini dikumpulkan menjadi beberapa lapisan yang terbagi menjadi *Input Layer* (lapisan masukan), *Hidden Layer* (lapisan tersembunyi karena berada di tengah), dan *Output Layer* (lapisan keluaran). Lapisan yang berbeda dapat melakukan perubahan yang berbeda pada *input*-nya tergantung dari fungsi non-linier yang diterapkan (Hardesty, 2017).

Artificial Neural Network belajar (atau dilatih) dengan memproses data, yang masing-masing berisi *input* dan *output* yang diketahui, sehingga membentuk asosiasi bobot dan probabilitas diantara keduanya. Asosiasi ini kemudian disimpan dalam struktur data *Artificial Neural Network* itu sendiri. Pelatihan *Artificial Neural Network* dilakukan dengan menentukan perbedaan antara *output* yang diprediksi dan *output target*. Perbedaan ini disebut sebagai kesalahan yang mana ingin diminimalisir melalui proses pembelajaran ini. *Artificial Neural Network* kemudian menyesuaikan bobotnya berdasarkan nilai kesalahan ini. Penyesuaian berturut-turut akan menyebabkan *Artificial Neural Network* menghasilkan *output* yang semakin mirip dengan *output target*. Setelah penyesuaian dalam jumlah yang dirasa cukup, pelatihanpun

dapat dihentikan dan *Artificial Neural Network* dianggap bisa memberikan prediksi yang tepat (Hardesty, 2017).

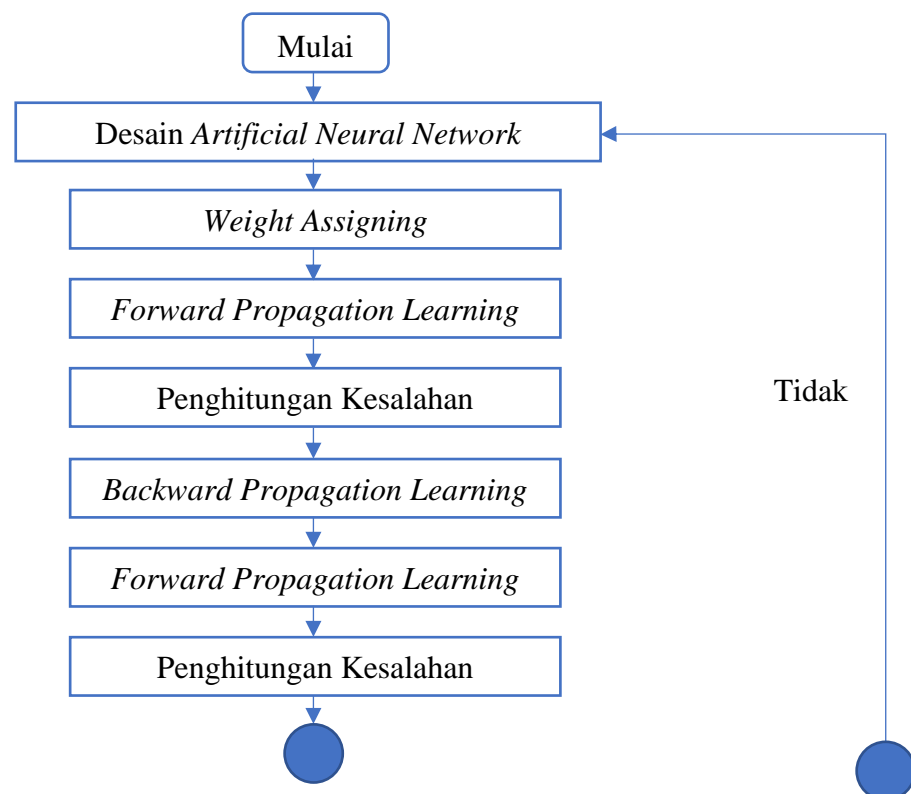


Gambar 2.4 *Artificial Neural Network* dengan 4 Lapisan (Larasati, 2019)

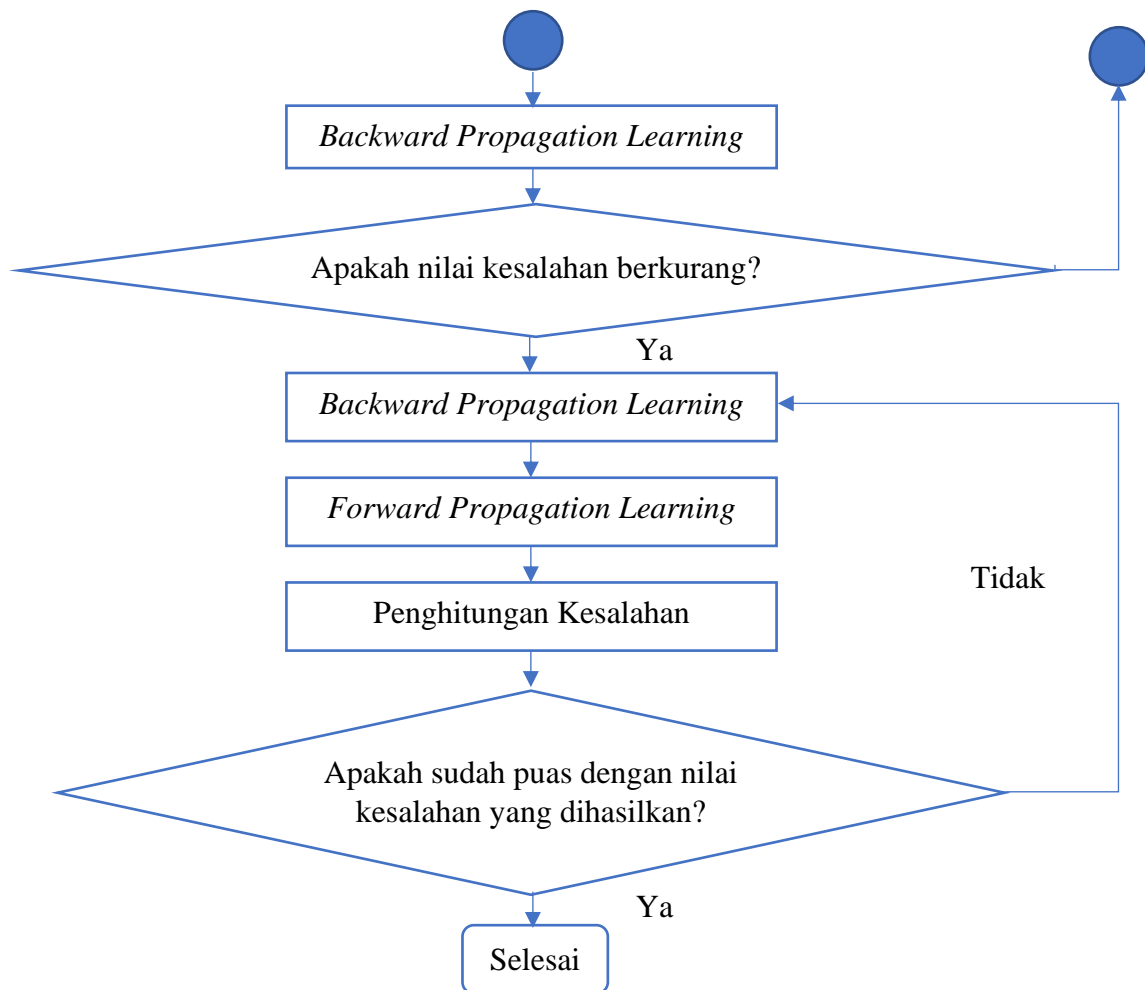
Proses kerja dari *Artificial Neural Network* terdiri dari beberapa langkah yakni:

1. Desain dimana penulis mendesain *Artificial Neural Network* yang sesuai untuk digunakan dalam pemecahan masalah penulis. Penulis menentukan ada berapa banyak lapisan *Input Layer* *Hidden Layer* dan *Output Layer* yang dibutuhkan serta ada berapa banyak neuron buatan untuk tiap lapisan tersebut. Penulis bisa mencari tahu berapa banyak neuron buatan yang diperlukan untuk *Input Layer* dengan cara mencari tahu berapa banyak masukan data yang ingin dipelajari oleh *Artificial Neural Network*. Untuk lapisan *Output Layer*, dengan konsep yang sama, perlu penulis cari tahu berapa banyak keluaran yang ingin dihasilkan oleh *Artificial Neural Network*. Untuk *Hidden Layer*, jumlahnya penulis disesuaikan sesuai dengan perjalanan sinyal dari *Input Layer* menuju *Output Layer*.
2. *Weight Assigning* dimana penulis memberikan bobot pada setiap Setiap neuron dan *edge* yang ada di dalam *Artificial Neural Network*. Hal ini penulis lakukan secara acak terlebih dahulu karena penulis pasti tidak akan tahu seberapa besar bobot yang harus penulis masukkan pada awalnya tanpa melakukan proses pembelajaran terlebih dahulu. Jika penulis sudah tahu harus memasukkan berapa bobot pada tiap neuron dan *edge* yang ada, maka sejatinya penulis tidak perlu menggunakan *Artificial Neural Network*.
3. *Forward Propagation Learning* dimana ini adalah proses adaptasi *Artificial Neural Network* untuk bisa menghasilkan prediksi yang lebih baik dengan mempertimbangkan sampel data yang sudah ada. Disebut sebagai *Forward Propagation* karena pada proses ini sinyal akan berjalan ke “depan” dari *Input Layer* menuju *Output Layer*. Bobot dari tiap neuron dan *edge* akan dilakukan kalkulasi menggunakan sebuah fungsi non-linier mulai dari *Input Layer* menuju *Output Layer*.
4. Penghitungan Kesalahan dimana pada saat nilai terakhir sampai kepada *Output Layer*, nilai tersebut akan dilakukan kalkulasi perbedaan antara nilai tersebut dengan *output target* yang benar sesuai dengan data yang dipelajari. Perbedaan ini disebut dengan kesalahan dan merupakan bagian penting dari pembelajaran *Artificial Neural Network*.

5. *Backward Propagation Learning* dimana nilai kesalahan di tarik ke “belakang” (dari *Output Layer* menuju *Input Layer*) untuk menyesuaikan bobot yang ada pada tiap neuron dan *edge*. Hal ini dilakukan agar *Artificial Neural Network* bisa belajar dari kesalahan tersebut dan bisa menyesuaikan bobot di tiap neuron dan *edge* sehingga bisa memberikan prediksi yang lebih baik.
6. Iterasi dimana dilakukan kembali *Forward Propagation Learning*, Penghitungan Kesalahan, dan *Backward Propagation Learning*. Hal ini dilakukan agar pada tiap iterasi, *Artificial Neural Network* dapat semakin baik menyesuaikan bobot dari tiap neuron dan *edge*-nya sehingga bisa semakin memperkecil perbedaan antara *output* yang diprediksi dan *output target* (memperkecil nilai kesalahan).
7. Evaluasi dimana penulis melihat apakah nilai kesalahan tiap iterasi dari *Forward Propagation Learning*, Penghitungan Kesalahan, dan *Backward Propagation Learning* memang memperkecil nilai kesalahan atau tidak. Jika nilai kesalahan tidak diperkecil dari tiap iterasi, maka penulis harus melakukan desain ulang dari *Artificial Neural Network* dan mengulang kembali proses belajar.
8. Terminasi dimana ketika penulis mendapati bahwa nilai kesalahan memang semakin mengecil setiap kali iterasi dari *Forward Propagation Learning*, Penghitungan Kesalahan, dan *Backward Propagation Learning* dan penulis sudah puas dengan perbedaan nilai kesalahan yang dihasilkan. Perlu diperhatikan bahwa nilai kesalahan dari *Artificial Neural Network* hampir tidak mungkin bernilai 0 sehingga penulis harus melakukan proses terminasi secara manual ketika dirasa pengurangan nilai kesalahan sudah tidak terlalu signifikan jika dibandingkan dengan usaha yang harus penulis lakukan saat melakukan pembelajaran. Nantinya *Artificial Neural Network* akan menghasilkan model yang bisa penulis gunakan secara langsung untuk pemrosesan data.



Gambar 2.5 Flowchart Cara Kerja Artificial Neural Network Bagian 1



Gambar 2.6 Flowchart Cara Kerja Artificial Neural Network Bagian 2

Artificial Neural Network dipilih untuk bisa menyelesaikan *Knapsack Problem* yang ditemui pada saat ingin melakukan *Cloud Provisioning* karena beberapa alasan, yaitu:

1. *Artificial Neural Network* dapat mempelajari, memproses, dan memprediksi hasil dari sebuah data. Dikarenakan penelitian ini akan menggunakan dataset, *Artificial Neural Network* merupakan pilihan yang cocok untuk digunakan.
2. *Artificial Neural Network* dapat menyesuaikan bobotnya berdasarkan nilai kesalahan yang dihasilkan dari tiap iterasi prosesnya. Hal ini akan sangat berguna ketika penulis ingin menghasilkan solusi berkualitas tinggi untuk optimasi penggunaan sumber daya *Cloud*.
3. *Artificial Neural Network* akan dipadukan bersama dengan *Genetic Algorithm*. Dimana *Artificial Neural Network* akan menjadi otak yang mempelajari, memproses, dan memprediksi hasil dari dataset yang ada dan melakukan *Cloud Provisioning* berdasarkan data tersebut. Sedangkan *Genetic Algorithm* akan digunakan untuk mengembangkan otak tersebut dan memastikan hanya *Artificial Neural Network* yang memiliki nilai *Fitness Function* yang tinggi yang akan berkembang menjadi generasi

berikutnya. Hal ini akan berakibat semakin bertambah tingginya solusi untuk optimasi penggunaan sumber daya *Cloud* pada tiap generasi baru (Suryansh, 2018).

4. Banyak iterasi dari *Artificial Neural Network* bisa disesuaikan dengan kebutuhan untuk mencapai nilai kesalahan minimal yang diinginkan untuk optimasi penggunaan sumber daya *Cloud*.
5. Karena kemampuannya untuk mereproduksi dan memodelkan banyak data, *Artificial Neural Network* telah diaplikasikan di banyak disiplin ilmu, termasuk pada optimasi penggunaan sumber daya.

2.2.7 CloudSim

CloudSim adalah kerangka kerja *Open Source*, yang digunakan untuk mensimulasikan infrastruktur dan layanan *Cloud Computing*. *CloudSim* dikembangkan oleh organisasi bernama CLOUDS Lab dan ditulis seluruhnya dalam bahasa *Java*. *CloudSim* digunakan sebagai sarana untuk mengevaluasi hipotesis sebelum pengembangan perangkat lunak sesungguhnya dengan mereproduksi tes dan hasilnya dalam sebuah simulasi lingkungan *Cloud Computing* (Laboratory, 2018).

Ambil contoh jika penulis ingin menerapkan aplikasi atau sebuah situs web di dalam *Cloud*. Penulis pasti ingin menguji layanan dan muatan yang dapat ditangani oleh produk tersebut. Penulis juga ingin menyesuaikan kinerjanya agar dapat mengatasi kemacetan sebelum penerapan sesungguhnya. Penulis dapat melakukan evaluasi tersebut melalui pengkodean simulasi lingkungan *Cloud Computing* dengan bantuan berbagai kelas fleksibel dan dapat diskalakan yang disediakan oleh *CloudSim* secara gratis (Laboratory, 2018).

Beberapa fitur penting yang dimiliki oleh *CloudSim* adalah:

1. Pusat data, server, dan host yang tervirtualisasi dalam skala besar.
2. Kebijakan yang dapat disesuaikan untuk melakukan *Cloud Provisioning*.
3. Sumber daya *Cloud Computing* yang dapat menghitung penggunaan energi.
4. Dilengkapi dengan topologi jaringan pusat data dan aplikasi pengiriman pesan.
5. Bisa memasukkan Sumber daya *Cloud* secara dinamis ke dalam simulasi.
6. Kebijakan *Cloud Provisioning* yang dapat ditentukan oleh pengguna.



Gambar 2.7 Melbourne CLOUDS Lab Pengembang CloudSIM (Laboratory, 2018)

2.2.8 Library CloudSim

CloudSim adalah kerangka kerja *Open Source* yang ditulis menggunakan bahasa pemrograman *Java* dan kelasnya terstruktur dengan cara yang sangat spesifik. Sehingga sangat penting bagi penulis untuk bisa memahami bagaimana arsitektur *Cloudsim* dibagi menjadi beberapa paket dan kelas yang memfasilitasi simulasi *Cloud* menggunakan *CloudSim* (SinghAnupinder, 2019).

Ada 12 *Namespace* dalam arsitektur *CloudSim* dimana setiap *Namespace* memiliki satu set kelas dengan fungsi spesifik yang berkorelasi dengan cara simulasi *Cloud* dijalankan. Ke-12 *Namespace* tersebut adalah (SinghAnupinder, 2019):

Tabel 2.2 *Namespace CloudSim*

No	Namespace	Fungsi
1	<i>Org.cloudbus.cloudsim</i>	<i>Namespace</i> ini berisi kelas model dari berbagai komponen perangkat keras dasar, kelompoknya, dan metode alokasi atau penggunaannya di <i>CloudSim</i> . Di sini, model berarti bahwa kelas ini berisi implementasi berbagai atribut dan perilaku komponen perangkat keras sistem <i>Cloud Computing</i> di dalam kehidupan nyata sebagai kumpulan metode <i>Java</i> . Setelah kelas-kelas ini dimulai selama simulasi, mereka akan mensimulasikan perilaku komponen sistem <i>Cloud Computing</i> nyata.
2	<i>Org.cloudbus.cloudsim.core</i>	<i>Namespace</i> ini berisi implementasi mesin simulasi, di mana kelas <i>cloudsim.java</i> yang ada di dalamnya adalah kelas utama dan bertanggung jawab untuk memulai dan menghentikan proses simulasi. Selain itu, terdapat kelas <i>simentity.java</i> yang berfungsi untuk mempertahankan status komponen <i>Cloud</i> yang disimulasikan. Terakhir, terdapat juga kelas <i>Simevent.java</i> , <i>futurequeue.java</i> , dan <i>defferedqueue.java</i> yang bertanggung jawab untuk memelihara panggilan peristiwa terkait komponen <i>Cloud</i> selama proses simulasi.
3	<i>Org.cloudbus.cloudsim.core.predicates</i>	<i>Namespace</i> ini berisi kelas-kelas yang terkait dengan pemanggilan peristiwa dari <i>defferedqueue.java</i> agar bisa diproses selama pemrosesan event simulasi.

No	Namespace	Fungsi
4	<i>Org.cloudbus.cloudsim.lists</i>	<i>Namespace</i> ini berisi implementasi daftar yang akan digunakan secara global selama proses simulasi. <i>Namespace</i> ini adalah kumpulan kelas khusus, di mana operasi penyortiran dan perbandingan diimplementasikan. Ada kelas daftar untuk host, elemen pemrosesan, tugas, mesin virtual, dan sumber daya.
5	<i>Org.cloudbus.cloudsim.power</i>	<i>Namespace</i> ini berisi implementasi komponen sistem <i>Cloud Computing</i> yang diperluas, yang dapat mensimulasikan skenario komputasi hijau (<i>Green Computing</i>) atau sadar daya (<i>Power Aware</i>).
6	<i>Org.cloudbus.cloudsim.power.lists</i>	<i>Namespace</i> ini hanya berisi implementasi <i>powervmlist.java</i> , yang merupakan versi lanjutan dari class <i>VmList.java</i> dari <i>namespace org.cloudbus.cloudsim.list</i>
7	<i>Org.cloudbus.cloudsim.power.models</i>	<i>Namespace</i> ini berisi kumpulan kelas yang menentukan konfigurasi daya server sebagai kelas model. Kelas-kelas ini meniru cara kerja sebenarnya dari berbagai merek mesin server yang tersedia di pasar dan membantu dalam menentukan konsumsi daya oleh mesin tersebut selama proses simulasi.
8	<i>Org.cloudbus.cloudsim.provisioners</i>	<i>Namespace</i> ini berisi implementasi perilaku tentang bagaimana komponen sistem <i>Cloud Computing</i> dapat disediakan.
9	<i>Org.cloudbus.cloudsim.util</i>	<i>Namespace</i> ini berisi implementasi fungsi perhitungan penting dan kumpulan utilitas umum.

CloudSim dipilih untuk bisa menyediakan simulasi lingkungan *Cloud Computing* untuk dilakukan penelitian *Cloud Provisioning* karena beberapa alasan, yaitu:

1. Open source dan bebas biaya, sehingga menguntungkan pengguna.
2. Mudah untuk diunduh dan diatur.
3. Lebih ramah ke pengguna dan dapat diperluas untuk mendukung pemodelan dan eksperimen.
4. Tidak memerlukan komputer dengan spesifikasi tinggi untuk bekerja.
5. Menyediakan kebijakan *Cloud Provisioning* yang telah ditentukan sebelumnya untuk mengelola sumber daya, dan juga memungkinkan penerapan algoritma yang ditentukan oleh pengguna.
6. Dokumentasi yang lengkap memberikan contoh yang telah dikodekan sebelumnya agar pengguna baru bisa terbiasa dengan berbagai fungsi dasar.

2.2.9 Library Encog

Encog adalah *framework* pembelajaran mesin yang tersedia untuk bahasa *Java* dan *.Net*. *Encog* mendukung berbagai algoritma pembelajaran mesin seperti *Bayesian Networks*, *Hidden Markov Models*, dan *Support Vector Machines*. Namun, kekuatan utamanya terletak pada algoritma *Artificial Neural Network*-nya. *Encog* berisi berbagai macam kelas untuk membuat berbagai macam *Artificial Neural Network*, serta memiliki kelas untuk normalisasi dan pemrosesan data untuk *Artificial Neural Network* ini (Heaton, *Encog: Library of Interchangeable Machine Learning Models*, 2015).

Encog memungkinkan pengguna *Java* untuk bereksperimen dengan berbagai macam model bahasa mesin menggunakan antarmuka yang sederhana dan konsisten untuk pengelompokan, regresi, dan klasifikasi. Hal ini memungkinkan pengguna untuk menemukan model mana yang paling cocok untuk data yang mereka gunakan. *Encog* menyediakan alat dasar untuk pembuatan model dan pengguna bisa langsung membuat model yang mereka inginkan berdasarkan spesifikasi yang sudah ditentukan sebelumnya (Heaton, *Encog: Library of Interchangeable Machine Learning Models*, 2015).



Gambar 2.8 Logo *Encog* (Heaton, *Encog Machine Learning Framework*, 2022)

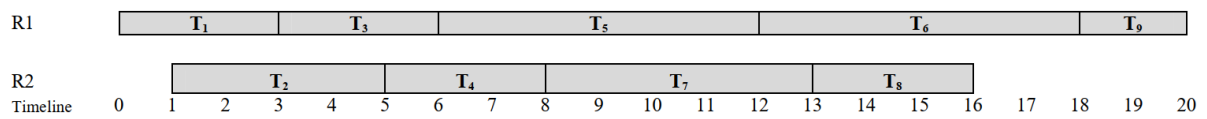
Encog dipilih untuk melakukan pengkodean *Artificial Neural Network* untuk dilakukan penelitian dalam *Cloud Provisioning* ini karena beberapa alasan, yaitu:

1. Semua model *Encog* diimplementasikan menggunakan algoritma *Multithreading* yang efisien, sehingga bisa mengurangi waktu pemrosesan data (Heaton, *Encog: Library of Interchangeable Machine Learning Models*, 2015).
2. *Encog* memiliki banyak fungsi pelatihan yang bisa dipilih untuk memaksimalkan akurasi yang dimiliki oleh model sehingga tidak terjadi *Overfitting* dan *Underfitting*.
3. *Encog* memiliki kelas bawaan untuk normalisasi data sehingga tidak perlu menggunakan *Library* lain untuk melakukan normalisasi.
4. Dokumentasi yang lengkap memberikan contoh yang telah dikodekan sebelumnya agar pengguna baru bisa terbiasa dengan berbagai fungsi dasar.

2.2.10 Definisi Parameter Penggunaan Sumber Daya Cloud

Untuk bisa mengetahui perbandingan efisiensi antara satu algoritma penjadwalan tugas dan alokasi mesin virtual (VM) dengan yang lainnya dalam meningkatkan tingkat penggunaan sumber daya *Cloud*, maka penulis perlu terlebih dahulu tahu parameter apa saja yang akan digunakan untuk bisa mengukurnya serta definisi dan cara menghitung parameter tersebut. Berikut adalah parameter yang akan digunakan dalam mencari tahu tingkat penggunaan sumber

daya *Cloud* (contoh akan dihitung berdasarkan dari gambar 2.7) (Henning Titi Ciptaningtyas, 2022):



Gambar 2.9 Contoh Penjadwalan Tugas (Henning Titi Ciptaningtyas, 2022)

Tabel 2.3 Parameter Tingkat Penggunaan Sumber Daya *Cloud*

Nama	Definisi	Cara Menghitung
<i>Makespan</i>	Waktu penyelesaian tugas terakhir (Henning Titi Ciptaningtyas, 2022).	$\max_{i \in \text{tasks}} \{F_i\}$ Contoh : 20 unit waktu karena T9 yang merupakan tugas terakhir selesai di unit ke-20
<i>Average Start Time</i>	Waktu rata-rata tugas mulai dieksekusi (Pradeep Singh Rawat, 2020).	$\frac{\sum_{i=1}^n \text{Waktu Ri Mulai}}{nR}$ Contoh: $(0+1)/2 = 0.5$
<i>Average Finish Time</i>	Waktu rata-rata tugas selesai dieksekusi (Pradeep Singh Rawat, 2020).	$\frac{\sum_{i=1}^n \text{Waktu Ri Selesai}}{nR}$ Contoh: $(20+16)/2 = 18$
<i>Average Execution Time</i>	Rata-rata lama waktu yang dibutuhkan untuk tugas selesai dieksekusi (Pradeep Singh Rawat, 2020).	$\frac{\sum_{i=1}^n \text{Waktu Ti}}{nT}$ Contoh: $(3+3+6+6+2+4+3+5+3)/9 = 32.3$
<i>Total Wait Time</i>	Lama waktu yang dibutuhkan untuk tugas pertama mulai dieksekusi (Pradeep Singh Rawat, 2020).	Contoh : Dari gambar bisa diambil bahwa <i>Total Wait Time</i> adalah 0, karena tidak ada delay untuk penjadwalan tugas
<i>Scheduling Length</i>	Jumlah total waktu yang dibutuhkan dari awal simulasi dimulai hingga simulasi selesai (Farouk A. Emara, 2021).	$\text{Scheduling Time} + \text{Makespan}$ Contoh: $0+20 = 20$
<i>Throughput</i>	Jumlah total tugas yang menyelesaikan eksekusi per unit waktu (Henning Titi Ciptaningtyas, 2022).	$\frac{nT}{\text{Makespan}}$ Contoh: $9/20 = 0.45$
<i>Resource Utilization</i>	Persentase penggunaan sumber daya saat pengerjaan tugas (Henning Titi Ciptaningtyas, 2022).	$\frac{\sum_{i=1}^n \text{Waktu Ri Selesai}}{\text{Makespan} * nR}$ Contoh: $(20+15) / (20*2) = 0.875$

Nama	Definisi	Cara Menghitung
<i>Energy Consumption</i>	Jumlah total penggunaan energi saat pengerjaan tugas (Farouk A. Emara, 2021).	
<i>Imbalance Degree</i>	Pengukuran ketidakseimbangan diantara semua pusat data (Farouk A. Emara, 2021).	$\frac{ET_{max} - ET_{min}}{ET_{avg}}$ <p>Dimana ET_{max}, ET_{min}, ET_{avg} adalah <i>Execution Time</i> maksimum, minimum, dan rata-rata secara urut dari semua pusat data yang ada.</p> <p>Contoh: $(6-2)/32.3 = 0.12$</p>

2.2.11 Tingkatan Penggunaan Sumber Daya

Tingkatan pemanfaatan sumber daya membantu penulis untuk memahami bagaimana sistem penulis menghabiskan waktu dan tenaga untuk bisa menyelesaikan tugas yang penulis minta, sehingga penulis dapat membuat keputusan manajemen sumber daya yang lebih efisien yang dapat memaksimalkan produktivitas dan profitabilitas (Meier, 2020). Hal ini dikarenakan pemanfaatan yang berlebihan (misalnya, bekerja lebih dari waktu dan tenaga yang tersedia) dapat menyebabkan penurunan performa sistem penulis. Sedangkan pemanfaatan yang kurang (misalnya, bekerja kurang dari waktu dan tenaga yang tersedia) dapat menyebabkan penundaan penyelesaian tugas (Education, 2021). Lalu berapakah persentase tingkatan pemanfaatan sumber daya yang direkomendasikan dan dianggap paling efisien? Menurut analisis *Gartner*, jawabannya adalah 70% hingga 80% (Moore, 2019).

2.2.12 Dataset The San Diego Supercomputer Center (SDSC) Blue Horizon Log

Dataset *San Diego Supercomputer Center (SDSC) Blue Horizon Log* adalah dataset yang akan digunakan untuk mensimulasikan permintaan sumber daya ke dalam simulasi sistem *Cloud Computing* dalam penelitian ini. *San Diego Supercomputer Center (SDSC) Blue Horizon Log* adalah sebuah catatan ekstensif yang dimulai saat mesin *Cloud Computing* baru saja dipasang di lab *San Diego Supercomputer Center (SDSC)*, dan kemudian mencatat catatan penggunaan mesin tersebut selama lebih dari dua tahun penggunaan produksi. Catatan ini berisi informasi tentang waktu saat *Node* diminta, waktu saat *Node* digunakan, waktu *CPU*, waktu pengiriman, waktu tunggu, waktu jalan, dan informasi tentang penggunaanya (log, 2003).

Ada 144 *Node* di dalam mesin *Cloud Computing* lab *San Diego Supercomputer Center (SDSC)*. Masing-masing adalah *SMP* 8 arah dengan palang yang menghubungkan prosesor ke memori bersama. Catatan ini mencatat penggunaan mesin tersebut mulai dari April 2000 hingga Januari 2003. Sistem penjadwalan tugas yang digunakan pada mesin ini disebut *Catalina*. Sistem ini dikembangkan di lab *San Diego Supercomputer Center (SDSC)*. Sistem ini menggunakan antrian prioritas, melakukan *Backfilling*, dan mendukung penggunaan reservasi (log, 2003).

2.2.13 Eclipse IDE

Eclipse adalah lingkungan pengembangan terintegrasi (IDE) yang digunakan dalam pemrograman komputer. *Eclipse* berisi ruang kerja dasar dan memiliki sistem *plug-in* yang dapat diperluas untuk menyesuaikan lingkungan pemrograman komputer. *Eclipse* adalah IDE paling populer kedua untuk pengembangan *Java*, dan, hingga 2016, adalah yang paling populer. *Eclipse* sebagian besar ditulis dalam bahasa *Java* dan penggunaan utamanya adalah untuk mengembangkan aplikasi berbasis *Java* (Foundation, 2001).

Eclipse memiliki peralatan pengembangan perangkat lunak (SDK) yang dimaksudkan untuk para pengembang *Java* namun tidak terbatas pada Bahasa *Java* saja. Pengguna dapat memperluas kemampuan *Eclipse* dengan memasang berbagai macam *plug-in* yang ditulis untuk *Eclipse Platform*, seperti peralatan pengembangan lunak untuk bahasa pemrograman lain. Pengguna bahkan dapat menulis dan menyumbangkan modul *plug-in* mereka sendiri. Peralatan pengembangan perangkat lunak *Eclipse* ini adalah perangkat lunak *open source* dan gratis sehingga bebas untuk digunakan tanpa perlu mengeluarkan biaya apapun (Foundation, 2001).



Gambar 2.10 Logo *Eclipse IDE* (Foundation, 2001)

BAB III METODOLOGI

3.1 Tahapan Metodologi Penelitian

Pada sub bab ini akan dibahas mengenai metodologi yang akan digunakan dalam pengerjaan tugas akhir. Metodologi yang digunakan dalam penelitian ini terdiri dari enam bagian antara lain:

1. Identifikasi Permasalahan

Pada bagian ini akan dilakukan pemahaman empatik atau pengidentifikasian masalah yang ingin dipecahkan.

2. Studi Literatur

Pada bagian ini akan dilakukan penentuan masalah inti dengan cara mencari referensi dari beberapa jurnal penelitian ataupun konferensi yang pernah dilakukan sebelumnya.

3. Analisis dan Desain Perangkat Lunak

Pada bagian ini akan ditetapkan ide yang diusulkan dalam bentuk desain purwarupa.

4. Implementasi Perangkat Lunak

Pada bagian ini akan dilakukan visualisasi dari ide yang diusulkan atau contoh dari purwarupa sistem penjadwalan tugas dan alokasi mesin virtual (VM).

5. Pengujian dan Evaluasi

Pada bagian ini dilakukan pengujian purwarupa berupa pengukuran parameter efisiensi sistem penjadwalan tugas dan alokasi mesin virtual (VM).

6. Penyusunan Buku Tugas Akhir

Pada Bagian ini akan dilakukan penyusunan buku tugas akhir yang merupakan laporan secara lengkap mengenai tugas akhir yang telah dikerjakan baik dari sisi teori, rancangan, maupun implementasi sehingga memudahkan bagi pembaca dan juga pihak yang ingin mengembangkan lebih lanjut

3.2 Deskripsi Metodologi Penelitian

Berikut ini merupakan tahapan metode yang akan digunakan oleh penulis dalam penelitian tugas akhir ini. Bagian ini penting untuk memahami apa saja tahapan dan proses yang akan dilakukan oleh peneliti dalam menjalankan penelitian ini.

3.2.1 Identifikasi Permasalahan

Pada tahap ini akan dilakukan kegiatan mencari permasalahan yang ada di sekitar penulis ataupun terinspirasi dari jurnal penelitian. Setelah permasalahan ditemukan, langkah selanjutnya adalah mencari solusi yang dapat digunakan untuk menyelesaikan permasalahan tersebut. Hasil dari tahap ini merupakan permasalahan dan usulan solusi yang dapat diangkat menjadi topik tugas akhir.

3.2.2 Studi Literatur

Tugas akhir ini menggunakan beberapa literatur yang sudah pernah dibuat sebelumnya seperti “*Genetic-Based Multi-objective Task Scheduling Algorithm in Cloud Computing Environment*” (Farouk A. Emara, 2021), “*Resource provisioning in scalable cloud using bio-inspired artificial neural network model*” (Pradeep Singh RawatPriti, 2020), “*A Call for Energy Efficiency in Data Centers*” (Michael Pawlish A. S., 2014), dan “*Survey on Task Scheduling Methods in Cloud RPS System*” (Henning Titi Ciptaningtyas, 2022).

3.2.3 Analisis dan Desain Perangkat Lunak

Langkah -langkah dari analisis dan desain perangkat lunak yang akan dibuat adalah, pertama-tama melakukan analisa dan *preprocessing* pada dataset yang akan digunakan. Hasil dari *preprocessing* pada dataset ini akan menghasilkan dataset yang bersih dan memiliki format data yang sesuai sehingga siap untuk digunakan dalam penelitian.

Pada penelitian ini sendiri akan dijalankan dua skenario dimana untuk skenario pertama akan dilakukan penjadwalan tugas dan alokasi mesin virtual (VM) menggunakan *Genetic Algorithm* saja dan untuk skenario kedua akan dilakukan menggunakan *Genetic Algorithm* bersamaan dengan *Artificial Neural Network*. Kedua skenario ini akan menggunakan dua dataset seperti yang tertera pada Tabel 3.1 sebagai sumber data simulasi skenario penjadwalan tugas dan alokasi mesin virtual (VM). Nantinya kedua skenario ini akan dilakukan perbandingan untuk mencari tahu skenario mana yang lebih efisien.

Tabel 3.1 Dataset yang Digunakan

No	Nama Algoritma	Dataset 1	Dataset 2
1	<i>Genetic Algorithm</i>	Dataset yang dibuat sendiri	Dataset <i>The San Diego Supercomputer Center (SDSC) Blue Horizon logs</i> (log, 2003).
2	<i>Genetic Algorithm + Artificial Neural Network</i>	Dataset yang dibuat sendiri	Dataset <i>The San Diego Supercomputer Center (SDSC) Blue Horizon logs</i> (log, 2003).

Untuk skenario pertama, akan dilakukan percobaan menggunakan *Genetic Algorithm* saja. Pertama-tama akan dilakukan inisialisasi populasi awal dari jadwal sebagai pedoman pengalokasian *Task* kepada mesin virtual (VM) yang sesuai menggunakan *Genetic Algorithm* saja berdasarkan dataset yang digunakan. Untuk spesifikasi dari *Genetic Algorithm* yang akan digunakan pada percobaan ini bisa dilihat pada Tabel 3.2.

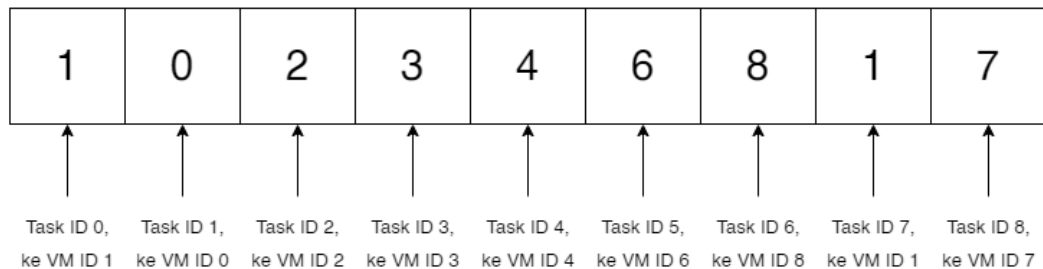
Tabel 3.2 Spesifikasi *Genetic Algorithm*

Dataset	Besar Populasi	Persentase Mutasi	Persentase Penyilangan	Jumlah Individu Elit	Banyak Loop
Dataset yang dibuat sendiri	20	30%	95%	2	20
Dataset <i>SDSC</i>	20	30%	95%	2	20

Spesifikasi ini akan mempengaruhi bagaimana *Genetic Algorithm* akan menghasilkan solusi jadwal yang efisien. Besar populasi adalah banyaknya jumlah individu yang berada di dalam satu populasi dalam satu generasi di dalam *Genetic Algorithm* (Mitchell, 1996).

Persentase mutasi adalah kemungkinan sebuah gen di dalam kromosom bisa secara acak berubah menjadi gen lain (Mitchell, 1996). Persentase penyilangan adalah kemungkinan diambilnya dua orangtua dari generasi sebelumnya untuk disilangkan gen-nya menjadi satu individu baru di generasi selanjutnya yang memiliki gabungan gen dari kedua orangtuanya (Mitchell, 1996). Sedangkan Jumlah individu elit adalah banyaknya individu yang memiliki nilai *Fitness Function* tinggi dari tiap generasi yang akan dimasukkan secara langsung sebagai anggota di generasi selanjutnya (Mitchell, 1996).

Jadwal ini sendiri akan berbentuk representasi kromosom dari individu di dalam *Genetic Algorithm* sepanjang 9 yang berisikan ID dari mesin virtual (VM) yang dituju oleh *Task*. Representasi kromosom ini bisa dilihat pada Gambar 3.1. Kromosom dari tiap individu di dalam populasi akan berisikan angka 0 (nol) hingga 8 (delapan) yang diisikan secara acak dengan duplikasi diperbolehkan. Angka-angka ini merepresentasikan ID Mesin Virtual yang akan menjadi tempat alokasi dari *Task*.



Gambar 3.1 Representasi Kromosom

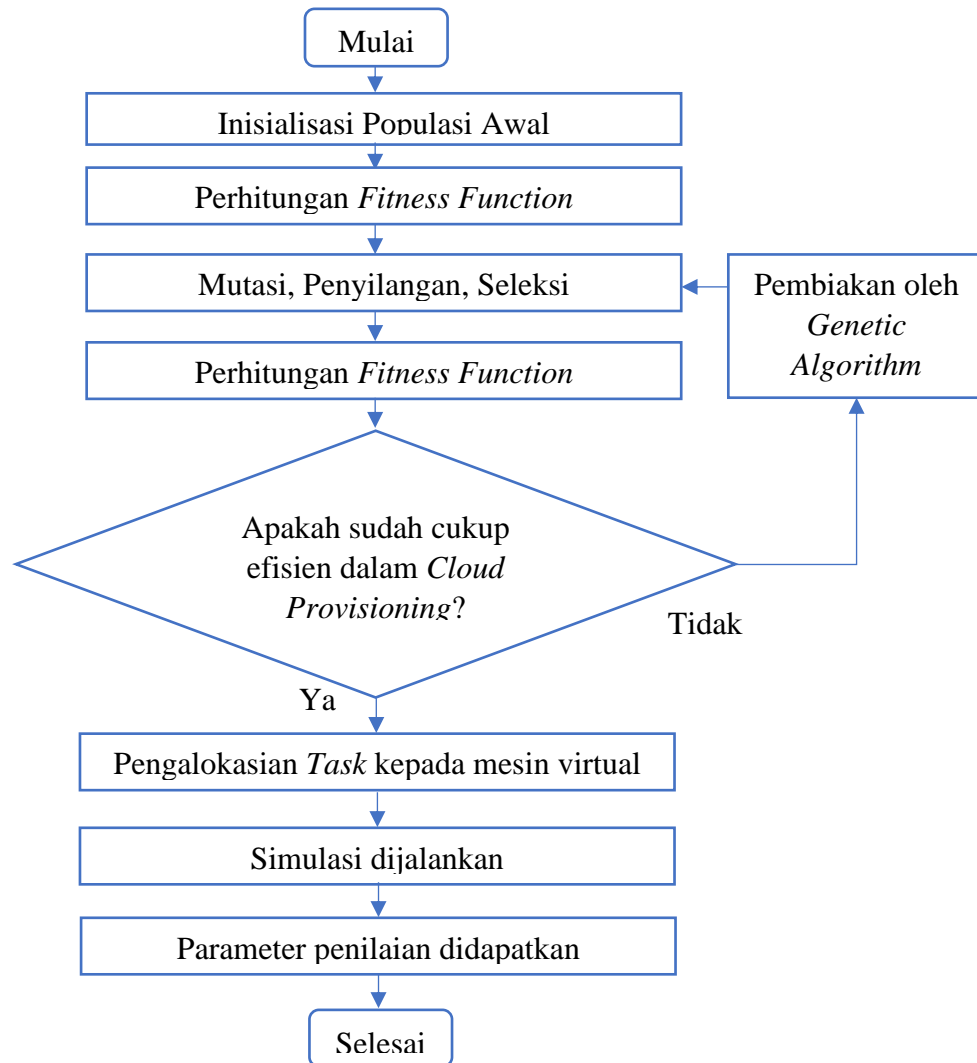
Jadwal ini akan dilakukan perhitungan *Fitness Function* untuk mencari tahu apakah mereka sudah cukup efisien dalam melakukan penjadwalan tugas dan alokasi mesin virtual (VM) atau belum berdasarkan *Fitness Function* yang bisa dilihat pada Tabel 3.3.

Tabel 3.3 *Fitness Function*

No	Nama	Penulisan
1	Fitness Function	$\alpha * \frac{1}{Total\ Waktu\ Eksekusi} + \beta * \frac{1}{Persentase\ Kegagalan}$
2	Total Waktu Eksekusi	$\sum_{i=1}^n \frac{Panjang\ Task\ i}{Mips\ VM\ i}$
3	Persentase Kegagalan	$\frac{jumlah\ task\ gagal}{total\ waktu}$

Disini Total Waktu Eksekusi berarti total waktu yang dihabiskan oleh *Task* di dalam eksekusi dan semakin kecil nilainya maka akan semakin baik sebuah solusi. Persentase Kegagalan berarti banyaknya *Task* yang gagal diproses per satu satuan waktu dan semakin kecil nilainya maka akan semakin baik sebuah solusi. α dan β adalah persentase seberapa penting nilai Total Waktu Eksekusi dan nilai Persentase Kegagalan dalam mempengaruhi kualitas solusi dan total nilai keduanya harus sama dengan 1. Rumus Total Waktu Eksekusi adalah jumlah total semua panjang *Task* dibagi dengan MIPS (*Million Instruction Per Second*) dari mesin virtual (VM) tempat dia akan diproses. Sedangkan rumus Persentase Kegagalan adalah banyaknya *Task* yang gagal diproses dibagi dengan lama waktu simulasi.

Jadwal-jadwal ini kemudian akan dilakukan seleksi, mutasi, dan penyilangan menggunakan *Genetic Algorithm* untuk menghasilkan generasi baru yang lebih efisien dari generasi sebelumnya dihitung dari *Fitness Function*-nya. Ketika dirasa tidak ada kenaikan yang signifikan dari *Fitness Function*-nya atau kondisi terminasi sudah tercapai, maka *Genetic Algorithm* akan diberhentikan dan jadwal yang memiliki *Fitness Function* tertinggi akan dijadikan keluaran dari skenario pertama ini. *Task* bisa dialokasikan kepada mesin virtual (VM) berdasarkan jadwal tersebut dan parameter penilaian bisa didapatkan dari hasil simulasinya. Skenario pertama ini bisa dilihat lebih detail pada Gambar 3.2 dibawah ini:



Gambar 3.2 Flowchart Skenario Pertama

Untuk skenario kedua, akan dilakukan percobaan kombinasi dari *Genetic Algorithm* dengan *Artificial Neural Network*. Pertama-tama hasil keluaran jadwal dari skenario pertama akan dilakukan pembagian menjadi 2 yaitu dataset pembelajaran sebanyak 80% dan dataset pengujian sebanyak 20% (Farouk A. Emara, 2021). Dataset pembelajaran akan dilakukan analisa dan dipelajari terlebih dahulu oleh *Artificial Neural Network* untuk menghasilkan model jadwal yang memiliki akurasi yang sesuai berdasarkan keluaran jadwal dari *Genetic Algorithm*. Spesifikasi dari *Artificial Neural Network* untuk penelitian ini bisa dilihat pada Tabel 3.4 dibawah ini.

Tabel 3.4 Spesifikasi *Artificial Neural Network*

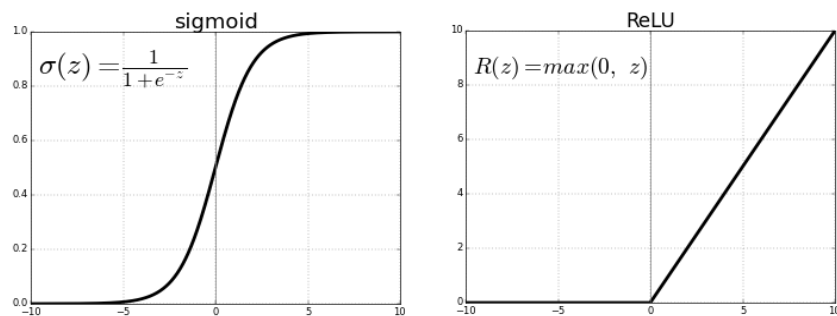
Dataset	Input Node	Hidden Node	Output Node	Metode Propagasi	Banyak Epoch	Akurasi
Dataset yang dibuat sendiri	9	18	9	<i>Manhattan</i>	100.000	88%
Dataset <i>SDSC</i>	9	18	9	<i>Manhattan</i>	100.000	88%

Spesifikasi 9 *Input Node* berguna untuk menerima masukan sebanyak 9 panjang *Task* yang nantinya akan menjadi informasi awal bagi *Artificial Neural Network* untuk mengeluarkan keluaran sebanyak 9 ID VM pada *Output Node*-nya. Di antara *Input Node* dan *Output Node* akan ada 1 *Hidden Node* berisi 18 *Node* sebagai penengah antara keduanya. *Hidden Node* ini akan menggunakan *Activation Function ReLu* dan *Output Node* akan menggunakan *Activation Function Sigmoid* (Heaton, Programming Neural Networks with Encog3 in Java, 2011). Lebih jelasnya bisa melihat pada Tabel 3.5 dibawah ini.

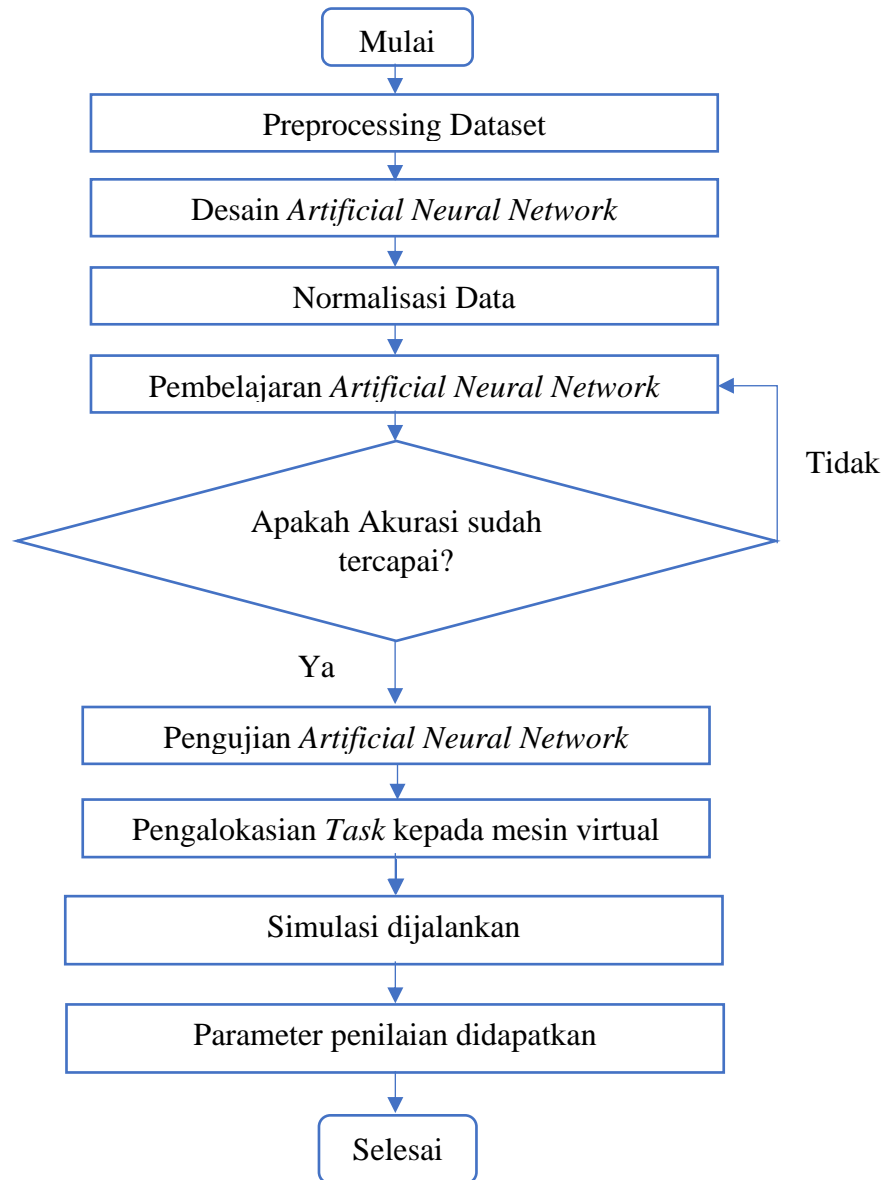
Tabel 3.5 Spesifikasi *Node Artificial Neural Network*

Dataset	Input	Output	Activation Function Pada Hidden Node	Activation Function Pada Output Node
Dataset yang dibuat sendiri	9 Panjang <i>Task</i>	9 ID VM	<i>ReLu</i>	<i>Sigmoid.</i>
Dataset <i>The San Diego Supercomputer Center (SDSC) Blue Horizon logs</i> (log, 2003).	9 Panjang <i>Task</i>	9 ID VM	<i>ReLu</i>	<i>Sigmoid.</i>

Propagasi *Manhattan* digunakan pada *Artificial Neural Network* ini agar penulis bisa mengontrol secara langsung seberapa besar pengurangan *weight* pada *Artificial Neural Network* sehingga penulis bisa mendapatkan tingkat akurasi yang sesuai. Hal ini dikarenakan Propagasi *Manhattan* menggunakan sebuah nilai yang sangat kecil untuk mengurangi ataupun menambahkan *weight*, tidak seperti metode propagasi biasa yang menggunakan *Gradient* sehingga perubahan nilai pada *weight* terlalu besar dan akurasi yang dihasilkan tidak bisa penulis kontrol (Heaton, Programming Neural Networks with Encog3 in Java, 2011). Akurasi yang penulis tuju pada percobaan ini adalah 88%, karena jika terlalu tinggi akan mengakibatkan *Overfit* dan jika terlalu kecil akan mengakibatkan *Underfit*.

Gambar 3.3 *Activation Function Sigmoid dan ReLu* (Sharma, 2017)

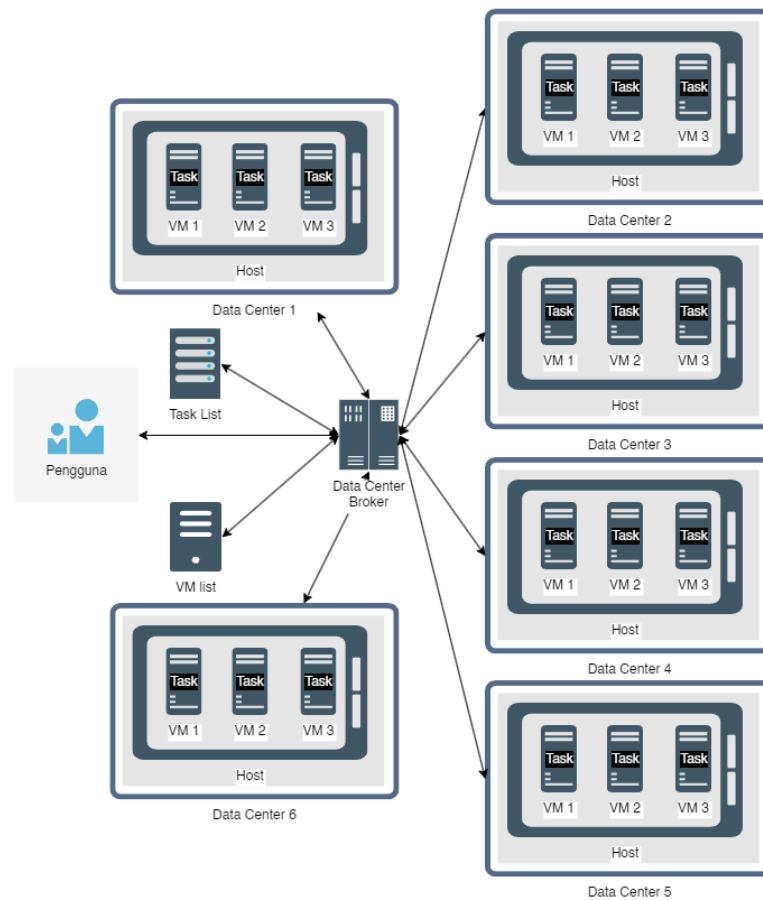
Setelah dataset pembelajaran dilakukan analisa dan dipelajari oleh *Artificial Neural Network*, model dari prediksi jadwal akan dihasilkan. Model ini akan dilakukan pengujian menggunakan dataset pengujian dan hasil dari jadwal yang dihasilkan akan dimasukkan kedalam simulasi di dalam *CloudSim*. *Task* bisa dialokasikan kepada mesin virtual (VM) berdasarkan jadwal tersebut dan parameter penilaian bisa didapatkan dari hasil simulasinya. Skenario kedua ini bisa dilihat lebih detail pada Gambar 3.4 dibawah ini:



Gambar 3.4 *Flowchart* Skenario Kedua

Setelah kedua skenario diatas selesai dijalankan, maka parameter penilaian dari kedua skenario tersebut akan dilakukan perbandingan untuk mencari tahu kombinasi algoritma mana yang lebih efisien dalam melakukan penjadwalan tugas dan alokasi mesin virtual (VM).

3.2.4 Implementasi Perangkat Lunak



Gambar 3.5 Skema Implementasi Perangkat Lunak



Gambar 3.6 Skema Implementasi Perangkat Lunak Pembagian VM

Implementasi dari perangkat lunak akan menggunakan *CloudSIM*. Sebuah kerangka kerja *Open Source*, yang digunakan untuk mensimulasikan layanan *Cloud Computing*. Perangkat lunak ini akan dilakukan implementasi kepada 54 *Virtual Machine* yang berada di dalam 18 *Host*, yang berada di dalam 6 *Data Center*. Masing-masing *Data Center* ini akan terhubung kepada sebuah *Data Center Broker* yang berguna sebagai otak penjadwalan tugas dan alokasi mesin virtual (VM). *Data Center Broker* ini akan terhubung ke *VM List* sebagai daftar *Virtual Machine* yang ada dan status mereka, ke *Task List* sebagai daftar *Task* yang ada dan status mereka, dan terakhir ke pengguna sebagai entitas yang memberikan *Task* dan menerima *Output* hasil pengerjaan. Skema implementasi ini bisa dilihat pada Gambar 3.5 dan Gambar 3.6.

Setiap *Virtual Machine* dan *Data Center* akan memiliki spesifikasi mereka masing-masing agar penulis bisa melihat hasil yang heterogen. Spesifikasi dari *Virtual Machine*, *Host*, dan *Data Center* ini bisa dilihat secara detail pada Tabel 3.6, Tabel 3.7, dan Tabel 3.8

Tabel 3.6 Spesifikasi *Virtual Machine*

ID	Memori (MB)	RAM (MB)	MIPS tiap Prosesor	Jumlah Prosesor	Bandwith (MBps)
VM 1	1000	512	400	1	1000
VM 2	1000	1024	500	1	1000
VM 3	1000	2048	600	1	1000

Tabel 3.7 Spesifikasi *Host*

ID	Memori (MB)	RAM (MB)	Jumlah Core	MIPS Core 1	MIPS Core 2	MIPS Core 3	MIPS Core 4	Max Power (W)	Static Power (%)
H1	1000000	128000	4	300	400	500	600	117	50
H2	1000000	128000	4	300	400	500	600	117	50
H3	1000000	128000	4	300	400	500	600	117	50

Tabel 3.8 Spesifikasi *Data Center*

ID	Jumlah Prosesor	Jumlah Host	Bandwith (MBps)	Latency (ms)
D1	6	3	10000	6
D2	6	3	10000	6
D3	6	3	10000	8
D4	6	3	10000	8
D5	6	3	10000	10
D6	6	3	10000	10

3.2.5 Pengujian dan Evaluasi

Pengujian dan evaluasi akan dilaksanakan dengan uji coba menggunakan simulasi *Cloud Environment* yang dijalankan pada *CloudSIM* untuk menguji efisiensi melakukan *Cloud Provisioning* menggunakan algoritma *Genetic Algorithm* dan *Genetic Algorithm* bersamaan dengan *Artificial Neural Network*. Kedua skenario tersebut kemudian akan dilakukan perbandingan untuk mencari tahu mana sistem *Cloud Provisioning* yang lebih efisien.

Perbandingan tersebut akan didasarkan pada beberapa parameter sesuai dengan Tabel 3.9 dibawah ini:

Tabel 3.9 Parameter yang Digunakan

No	Parameter	Rumus
1	<i>Makespan</i>	$\max_{i \in tasks} \{F_i\}$
2	<i>Average Start Time</i>	$\frac{\sum_{i=1}^n Waktu Ri Mulai}{nR}$
3	<i>Average Finish Time</i>	$\frac{\sum_{i=1}^n Waktu Ri Selesai}{nR}$
4	<i>Average Execution Time</i>	$\frac{\sum_{i=1}^n Waktu Ti}{nT}$
5	<i>Total Wait Time</i>	<i>Total Delay</i>
6	<i>Scheduling Length</i>	<i>Scheduling Time + Makespan</i>
7	<i>Throughput</i>	$\frac{nT}{Makespan}$
8	<i>Resource Utilization</i>	$\frac{\sum_{i=1}^n Waktu Ri Selesai}{Makespan * nR}$
9	<i>Energy Consumption</i>	<i>Total Energy Consumption</i>
10	<i>Imbalance Degree</i>	$\frac{ET_{max} - ET_{min}}{ET_{avg}}$

3.2.6 Penyusunan Buku Tugas Akhir

Pada tahap ini akan dilakukan penyusunan laporan yang menjelaskan dasar teori dan metode yang digunakan dalam tugas akhir ini serta hasil dari implementasi aplikasi perangkat lunak yang telah dibuat.

3.3 Jadwal Pengerjaan Tugas Akhir

Pada bagian ini akan dibahas mengenai rencana jadwal pengerjaan tugas akhir yang akan direpresentasikan pada tabel 3.10 dibawah ini:

Tabel 3.10 Jadwal Pengerjaan

Tahapan	2022																							
	Juli			Agustus			September			Oktober			November			Desember								
Pembuatan Proposal Tugas Akhir																								
Belajar CloudSIM Melalui Tutorial Online																								
Studi Literatur																								
Pembuatan Perangkat Lunak																								
Pengujian Perangkat Lunak																								
Evaluasi Perangkat Lunak																								
Penyusunan Buku Tugas Akhir																								

BAB IV

IMPLEMENTASI

4.1 Pengaturan Konfigurasi Simulasi Cloud Pada CloudSim

Agar simulasi pada penelitian ini bisa berjalan dengan lancar, diperlukan beberapa pengaturan konfigurasi pada *CloudSim*. Konfigurasi ini berupa melakukan pengaturan pada *Task*, pengaturan pada mesin virtual (VM), pengaturan pada *Host*, dan juga pengaturan pada *Data Center*.

4.1.1 Pengaturan Pada Task

Pengaturan pada *Task* dilakukan pada metode *createCloudlet* yang ada pada *Library CloudSim*. Pengaturan ini berupa penentuan parameter dari *Task* seperti panjang *Task*, besar *Task* saat *input*, besar *Task* saat *output*, berapa banyak *Core* yang dibutuhkan untuk menyelesaikan *Task*, dan model utilisasi *Task*. Pengaturan ini bisa dilihat lebih detail pada Kode Sumber 4.1 dibawah ini.

```
ArrayList<Double> randomSeed = getSeedValue(cloudlets);

// Creates a container to store Cloudlets
LinkedList<Cloudlet> list = new LinkedList<Cloudlet>();

//Cloudlet parameters
long length = 1000; // Cloudlet length (MI) - 1000 for Random Dataset
long fileSize = 300; // Cloudlet file size (MB)
long outputSize = 300; // Cloudlet file size (MB)
int pesNumber = 1; // Cloudlet CPU needed to process
UtilizationModel utilizationModel = new UtilizationModelFull();

Cloudlet[] cloudlet = new Cloudlet[cloudlets];

for(int i=0;i<cloudlets;i++)
{
    // Taking input from the dataset
    long finallen = length + Double.valueOf(randomSeed.get(i)).longValue();

    // Creating the cloudlet with all the parameter listed
    cloudlet[i] = new Cloudlet(i, finallen, pesNumber, fileSize, outputSize,
utilizationModel, utilizationModel, utilizationModel);

    // setting the owner of these Cloudlets
    cloudlet[i].setUserId(userId);
    list.add(cloudlet[i]);
}

return list;
```

Kode Sumber 4.1 Metode *createCloudlet*

4.1.2 Pengaturan Pada Mesin Virtual (VM)

Pengaturan pada mesin virtual (VM) dilakukan pada metode *createVM* yang ada pada *Library CloudSim*. Pengaturan ini berupa penentuan parameter dari VM seperti memori VM, RAM VM, MIPS tiap prosesor, jumlah prosesor, dan *Bandwith*. Pengaturan ini akan disesuaikan dengan Tabel 3.8. Pengaturan ini bisa dilihat lebih detail pada Kode Sumber 4.2 dibawah ini.

```
//Creates a container to store VMs.
//This list is passed to the broker later
LinkedList<Vm> list = new LinkedList<Vm>();

//VM Parameters
long size = 10000; //Image size (MB)
int[] ram = {512,1024,2048}; //VM memory (MB)
int[] mips = {400,500,600}; //VM processing power (MIPS)
long bw = 1000; //VM bandwidth
int pesNumber = 1; //Number of cpus
String vmm = "Xen"; //VMM name

//create VMs
Vm[] vm = new Vm[vms];

for(int i=0;i<vms;i++)
{
    //For loop to create a VM with a time shared scheduling policy for cloudlets:
    vm[i] = new Vm(i, userId, mips[i%3], pesNumber, ram[i%3], bw, size, vmm, new
CloudletSchedulerSpaceShared());
    list.add(vm[i]);
}

return list;
```

Kode Sumber 4.2 Metode *createVM*

4.1.3 Pengaturan Pada Host

Pengaturan pada *Host* dilakukan pada metode *createDataCenter* yang ada pada *Library CloudSim*. Metode ini memiliki dua fungsi sekaligus dimana penulis bisa mengatur *Host* sekaligus mengatur *Data Center* dengan *Host* tersebut di dalam satu metode. Untuk sub-bab ini akan dibahas kode sumber bagian pengaturan *Host* saja. Pengaturan ini berupa penentuan MIPS tiap *Core*, RAM *Host*, Memori *Host*, *Bandwith*, *Max Power*, dan *Static Power Percentage*. Pengaturan ini akan disesuaikan dengan Tabel 3.9. Pengaturan ini bisa dilihat lebih detail pada Kode Sumber 4.3 dibawah ini.

```
int mipsunused= 300; // Unused core, only 3 cores will be able to process Cloudlets
for this simulation
int mips1 = 400; // The MIPS Must be bigger than the VMs
int mips2 = 500;
int mips3 = 600;

// 3. Create PEs and add these into the list.
//for a quad-core machine, a list of 4 PEs is required:
peList1.add(new Pe(0, new PeProvisionerSimple(mips1))); // need to store Pe id and
MIPS Rating, Must be bigger than the VMs
```

```

peList1.add(new Pe(1, new PeProvisionerSimple(mips1)));
peList1.add(new Pe(2, new PeProvisionerSimple(mips1)));
peList1.add(new Pe(3, new PeProvisionerSimple(mipsunused)));
peList2.add(new Pe(4, new PeProvisionerSimple(mips2)));
peList2.add(new Pe(5, new PeProvisionerSimple(mips2)));
peList2.add(new Pe(6, new PeProvisionerSimple(mips2)));
peList2.add(new Pe(7, new PeProvisionerSimple(mipsunused)));
peList3.add(new Pe(8, new PeProvisionerSimple(mips3)));
peList3.add(new Pe(9, new PeProvisionerSimple(mips3)));
peList3.add(new Pe(10, new PeProvisionerSimple(mips3)));
peList3.add(new Pe(11, new PeProvisionerSimple(mipsunused)));

//4. Create Hosts with its id and list of PEs and add them to the list of machines
int ram = 128000 ; //Host memory (MB), Must be bigger than the VMs
long storage = 1000000; //Host storage (MB)
int bw = 10000; //Host bandwidth
int maxpower = 117; // Host Max Power
int staticPowerPercentage = 50; // Host Static Power Percentage

hostList.add(
    new PowerHostUtilizationHistory(
        hostId, new RamProvisionerSimple(ram),
        new BwProvisionerSimple(bw),
        storage,
        peList1,
        new VmSchedulerTimeShared(peList1),
        new PowerModelLinear(maxpower, staticPowerPercentage)));

hostId++;

```

Kode Sumber 4.3 Metode *createDataCenter* Bagian Pengaturan *Host*

4.1.4 Pengaturan Pada Data Center

Pengaturan pada *Host* dilakukan pada metode *createDataCenter* yang ada pada *Library CloudSim*. Metode ini memiliki dua fungsi sekaligus dimana penulis bisa mengatur *Host* sekaligus mengatur *Data Center* dengan *Host* tersebut di dalam satu metode. Untuk sub-bab ini akan dibahas kode sumber bagian pengaturan *Data Center* saja. Pengaturan ini berupa penentuan ada berapa banyak *Host* di dalam tiap *Data Center* dan nantinya jumlah prosesor, *Bandwidth*, dan juga latensi akan menyesuaikan dengan *Host*. Penulis juga bisa melakukan pengaturan *OS*, *VMM Name*, dan juga *Cost*, tetapi pengaturan ini tidak akan digunakan pada penelitian ini. Pengaturan ini akan disesuaikan dengan Tabel 3.10. Pengaturan ini bisa dilihat lebih detail pada Kode Sumber 4.4 dibawah ini.

```

// 5. Create a DatacenterCharacteristics object that stores the
// properties of a data center: architecture, OS, list of
// Machines, allocation policy: time- or space-shared, time zone
// and its price (G$/Pe time unit).
String arch = "x86";           // System architecture
String os = "Linux";           // Operating system
String vmm = "Xen";            // Name
double time_zone = 10.0;       // Time zone this resource located
double cost = 3.0;              // The cost of using processing in this resource
double costPerMem = 0.05;      // The cost of using memory in this resource

```



```

double costPerStorage = 0.1;    // The cost of using storage in this resource
double costPerBw = 0.1;        // The cost of using bw in this resource
LinkedList<Storage> storageList = new LinkedList<Storage>();

DatacenterCharacteristics characteristics = new DatacenterCharacteristics(
arch, os, vmm, hostList, time_zone, cost, costPerMem, costPerStorage, costPerBw);

// 6. Finally, we need to create a PowerDatacenter object.
PowerDatacenter datacenter = null;
try
{
    datacenter = new PowerDatacenter(name, characteristics, new
PowerVmAllocationPolicySimple(hostList), storageList, 9);
} catch (Exception e)
{
    e.printStackTrace();
}

return datacenter;

```

Kode Sumber 4.4 Metode *createDataCenter* Bagian Pengaturan *Data Center*

4.2 Pengaturan Parameter Penilaian

Agar hasil simulasi pada penelitian ini bisa dilakukan perbandingan, maka diperlukan adanya pengaturan pada parameter penilaian. Parameter penilaian yang akan digunakan pada penelitian ini telah dijabarkan pada Tabel 3.11 dan rumus perhitungannya telah dijelaskan pada Tabel 2.4, sehingga pada bagian ini akan dijelaskan hanya pada implementasi kodenya saja.

4.2.1 Makespan

Untuk *Makespan*, pertama-tama diisikan dengan nilai nol terlebih dahulu untuk memastikan tidak ada nilai dari *run* sebelumnya yang tersimpan. Kemudian ditambahkan dengan Waktu Selesai dari *Task* terakhir. Implementasi kode bisa dilihat pada Kode Sumber 4.5 dibawah ini.

```

double makespan =0.0;
double makespan_total = makespan + cloudlet.getFinishTime();
System.out.println("Makespan: " + makespan_total);

```

Kode Sumber 4.5 *Makespan*

4.2.2 Average Start Time

Untuk *Average Start Time*, pertama-tama diisikan dengan nilai nol terlebih dahulu untuk memastikan tidak ada nilai dari *run* sebelumnya yang tersimpan. Kemudian ditambahkan dengan Waktu Dimulai dari semua *Task* yang ada pada simulasi dan penulis bagi dengan jumlah *Task*. Implementasi kode bisa dilihat pada Kode Sumber 4.6 dibawah ini.

```

double totalStartTime =0.0;
for (int i = 0; i < size; i++) {
    totalStartTime = cloudletList.get(i).getExecStartTime();
}
double avgStartTime = totalStartTime/size;

```

Kode Sumber 4.6 *Average Start Time*

4.2.3 Average Finish Time

Untuk *Average Finish Time*, pertama-tama diisikan dengan nilai nol terlebih dahulu untuk memastikan tidak ada nilai dari *run* sebelumnya yang tersimpan. Kemudian ditambahkan dengan Waktu Selesai dari semua *Task* yang ada pada simulasi dan penulis bagi dengan jumlah *Task*. Implementasi kode bisa dilihat pada Kode Sumber 4.7 dibawah ini.

```
double totalTime = 0.0;
for (int i = 0; i < size; i++) {
    totalTime = cloudletList.get(i).getFinishTime();
}
double avgTAT = totalTime/size;
System.out.println("Average FinishTime: " + avgTAT );
```

Kode Sumber 4.7 *Average Finish Time*

4.2.4 Average Execution Time

Untuk *Average Execution Time*, pertama-tama diisikan dengan nilai nol terlebih dahulu untuk memastikan tidak ada nilai dari *run* sebelumnya yang tersimpan. Kemudian ditambahkan dengan Waktu Eksekusi dari semua *Task* yang ada pada simulasi dan penulis bagi dengan jumlah *Task*. Implementasi kode bisa dilihat pada Kode Sumber 4.8 dibawah ini.

```
double ExecTime = 0.0;
for (int i = 0; i < size; i++) {
    ExecTime = cloudletList.get(i).getActualCPUTime();
}
double avgExecTime = ExecTime/size;
System.out.println("Average Execution Time: " + avgExecTime );
```

Kode Sumber 4.8 *Average Execution Time*

4.2.5 Total Wait Time

Untuk *Total Wait Time*, *Library CloudSim* sudah memberikan variabel yang sudah menyimpan *Total Wait Time* sehingga penulis hanya perlu memanggilnya saja. Nama variabel ini adalah *waitTimeSum*. Implementasi kode bisa dilihat pada Kode Sumber 4.9 dibawah ini.

```
Log.println("TotalWaitTime : " + waitTimeSum);
```

Kode Sumber 4.9 *Total Wait Time*

4.2.6 Scheduling Length

Untuk *Scheduling Length*, tambahkan nilai dari *Total Wait Time* dan nilai dari *Makespan*. Implementasi kode bisa dilihat pada Kode Sumber 4.10 dibawah ini.

```
double scheduling_length = waitTimeSum + makespan_total;
Log.println("Total Scheduling Length: " + scheduling_length);
```

Kode Sumber 4.10 *Scheduling Length*

4.2.7 Throughput

Untuk *Throughput*, pertama-tama diisikan dengan nilai nol terlebih dahulu untuk memastikan tidak ada nilai dari *run* sebelumnya yang tersimpan. Kemudian dicari berapakah nilai terbesar dari *Finish Time* milik semua *Task* yang ada. Nilai ini nantinya akan menjadi pembagi dari banyaknya *Task* yang ada. Implementasi kode bisa dilihat pada Kode Sumber 4.11 dibawah ini.

```
double maxFT = 0.0;
for (int i = 0; i < size; i++) {
    double currentFT = cloudletList.get(i).getFinishTime();
    if (currentFT > maxFT) {
        maxFT = currentFT;
    }
}
double throughput = size/maxFT;
System.out.println("Throughput: " + throughput );
```

Kode Sumber 4.11 *Throughput*

4.2.8 Resource Utilization

Untuk *Resource Utilization*, penulis ambil waktu total yang semua *Task* habiskan di dalam proses. Kemudian, bagi waktu total tersebut dengan nilai *Makespan* dikalikan dengan jumlah *VM* yang mana adalah 54. Terakhir, kalikan nilainya dengan 100 untuk mengubah nilai tersebut menjadi persentase. Implementasi kode bisa dilihat pada Kode Sumber 4.12 dibawah.

```
double resource_utilization = (CPUTimeSum / (makespan_total * 54)) * 100;
Log.println("Resouce Utilization: " + resource_utilization);
```

Kode Sumber 4.12 *Resource Utilization*

4.2.9 Energy Consumption

Untuk *Energy Consumption*, *Library CloudSim* sudah memberikan metode yang sudah menyimpan *Total Energy Consumption* dari tiap *Data Center* sehingga penulis hanya perlu memanggilnya dan menambahkan nilai dari semua *Data Center*. Nama metode ini adalah *getPower*. Lalu nilai dari hasil penjumlahan ini dibagi dengan 3600 yang dikalikan 1000 untuk mengubah nilainya dalam satuan kWh. Implementasi kode bisa dilihat pada Kode Sumber 4.13 dibawah.

```
Log.println(String.format("Total Energy Consumption: %.2f kWh",
(datacenter1.getPower() + datacenter2.getPower()+ datacenter3.getPower()+
datacenter4.getPower()+ datacenter5.getPower()+ datacenter6.getPower())/
(3600*1000)));
```

Kode Sumber 4.13 *Energy Consumption*

4.2.10 Imbalance Degree

Untuk *Imbalance Degree*, penulis mengambil panjang *Task* terbesar menggunakan metode *getMax* dan mengurangnya dengan panjang *Task* terkecil menggunakan metode *getMin*. Nilai ini nantinya akan dibagi dengan waktu total yang semua *Task* habiskan di dalam proses

dibagi dengan jumlah semua panjang *Task* yang ada. Implementasi kode bisa dilihat pada Kode Sumber 4.14 dibawah.

```
double degreeimbalance = (stats.getMax() - stats.getMin())/(CPUTimeSum/ totalValues);
System.out.println("Imbalance Degree: " + degreeimbalance);
```

Kode Sumber 4.14 *Imbalance Degree*

4.3 Pembuatan Dataset Random

Salah satu sumber data pada penelitian ini adalah dataset *Random* yang dibuat sendiri oleh penulis. Dataset ini sebenarnya tidak dibuat penuh secara acak, dikarenakan apabila dibuat penuh secara acak, maka penulis tidak akan bisa memberikan lingkungan percobaan yang sama pada setiap percobaan yang dilakukan. Hal ini akan berakibat pada hilangnya status dataset *Random* sebagai variabel kontrol yang dipastikan harus sama pada setiap percobaan yang dilakukan. Oleh karena itu pembuatan dataset *Random* ini diperlukan cara khusus dimana tetap dibuat secara acak, namun tetap harus bisa direkreasi pada percobaan-percobaan selanjutnya. Pembuatan ini terbagi menjadi *Randomize* pada *Microsoft Excel*, membaca nilai dari *Microsoft Excel*, dan implementasi pada simulasi.

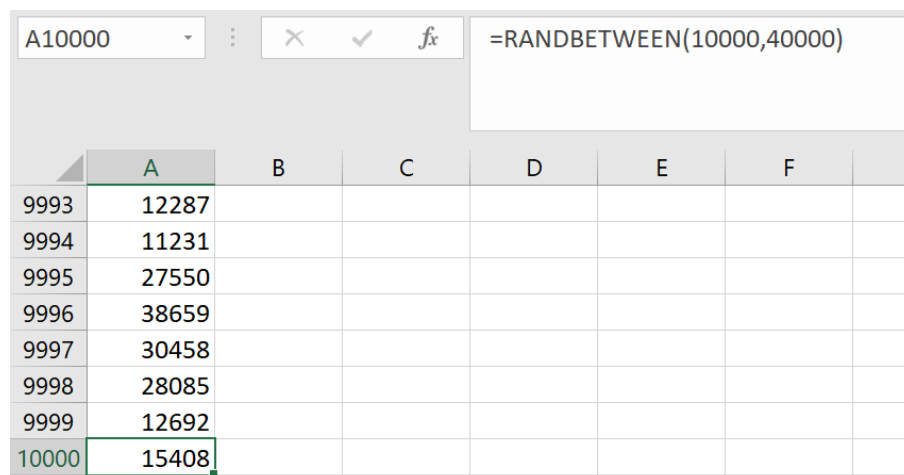
4.3.1 Randomize Pada Microsoft Excel

Microsoft Excel menyediakan metode untuk menghasilkan angka acak menggunakan metode *Randbetween*. Metode ini akan memberikan angka acak di antara 2 angka yang akan penulis definisikan (Bansal, 2017). Penulis mendefinisikan batas bawah angka acak adalah 10000 dan batas atas angka acak adalah 40000 sehingga penulis memasukkan Kode Sumber 4.14 pada *Microsoft Excel*.

```
Randbetween(10000,40000)
```

Kode Sumber 4.15 *Randbetween*

Lalu penulis menarik Kode Sumber 4.14 hingga pada baris 10000 untuk menghasilkan sebanyak 10000 nilai acak. Nilai-nilai acak ini nantinya akan berfungsi sebagai panjang *Task* yang akan dijalankan pada simulasi. Hasil pembuatan nilai-nilai ini bisa dilihat pada Gambar 4.1 dibawah ini.

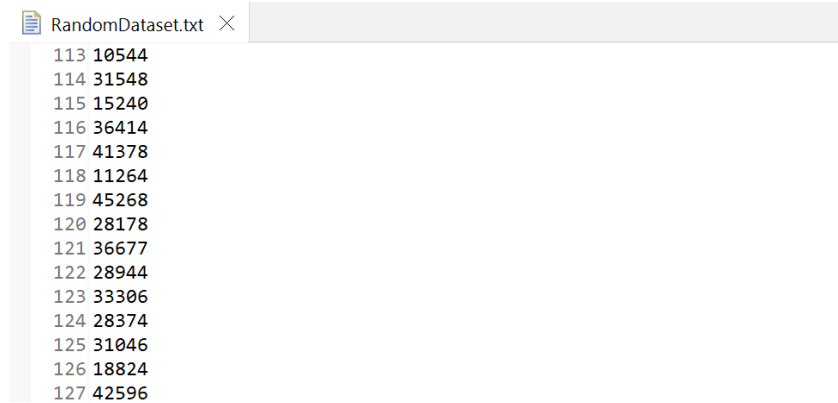


	A	B	C	D	E	F	G
9993	12287						
9994	11231						
9995	27550						
9996	38659						
9997	30458						
9998	28085						
9999	12692						
10000	15408						

Gambar 4.1 Hasil Pembuatan Nilai Acak

4.3.2 Membaca Nilai Dari Microsoft Excel

Karena Bahasa *Java* tidak bisa membaca nilai dari *Microsoft Excel* secara langsung, maka penulis memindahkan hasil pembuatan nilai acak dari Bagian 4.3.1 kepada *Notepad* yang penulis berikan nama *RandomDataset.txt* seperti bisa dilihat pada Gambar 4.2 dibawah. Nantinya Bahasa *Java* akan bisa membaca nilai dari file *txt* tersebut.



Gambar 4.2 Hasil Pemindahan Nilai Acak Pada *RandomDataset.txt*

Lalu penulis membuat sebuah metode di dalam *Java* untuk bisa membuka dan membaca nilai dari dataset tersebut sebagai nilai masukkan ke dalam simulasi *Cloud*. Nama dari metode ini adalah *getSeedValue*. Implementasi kode ini bisa dilihat pada Kode Sumber 4.15, dimana kode ini bekerja dengan membaca nilai dari *RandomDataset.txt* kemudian menyimpan nilai tersebut di dalam sebuah *ArrayList*. Metode ini akan membaca nilai secara terus-menerus hingga nilai dari *RandomDataset.txt* sudah habis, atau jumlah *Task* yang diminta sudah terpenuhi.

```
// Creating an arraylist to store Cloudlet Datasets
ArrayList<Double> seed = new ArrayList<Double>();
Log.println(System.getProperty("user.dir")+ "/RandomDataset.txt");

try{
    // Opening and scanning the file
    File fobj = new File(System.getProperty("user.dir")+ "/RandomDataset.txt");
    java.util.Scanner readFile = new java.util.Scanner(fobj);

    while(readFile.hasNextLine() && cloudletcount>0)
    {
        // Adding the file to the arraylist
        seed.add(readFile.nextDouble());
        cloudletcount--;
    }
    readFile.close();
} catch (FileNotFoundException e) {
    e.printStackTrace();
}

return seed;
```

Kode Sumber 4.16 *getSeedValue*

4.3.3 Implementasi Pada Simulasi

Untuk bisa mengimplementasikan nilai dari dataset tersebut pada simulasi, Kode Sumber 4.1, yaitu kode untuk pengaturan *Task* akan melakukan pemanggilan pada metode *getSeedValue*. Hal ini akan berakibat, Kode Sumber 4.1 memiliki nilai dari tersebut dan bisa menambahkan nilai tersebut kepada panjang *Task* yang dibuat sehingga bisa terimplementasi dengan sempurna pada simulasi *Cloud*.

4.4 Pembuatan Dataset SDSC Blue Horizon Logs

Salah satu sumber data lain pada penelitian ini adalah dataset dari *The San Diego Supercomputer Center (SDSC) Blue Horizon logs* (log, 2003). Untuk penelitian ini, penulis mengambil dataset dari versi *clean-4.2* yang ada pada *Website The San Diego Supercomputer Center (SDSC) Blue Horizon logs* (log, 2003). Pembuatan ini terbagi menjadi *Preprocessing* pada dataset, membaca nilai dari *Microsoft Excel*, dan implementasi pada simulasi.

4.4.1 Preprocessing Pada Dataset

Hasil unduhan dataset dari versi *clean-4.2* memberikan penulis dengan 18 kolom data dimana tiap kolom ini memiliki *class* nya masing-masing seperti bisa dilihat pada Gambar 4.3. Pada awalnya penulis ingin mengambil data dari kolom ke-10 yaitu Total Permintaan Memori Tiap Prosesor, hanya saja dataset ini menunjukkan angka -1 dari awal data hingga akhir, yang berarti tidak ada catatan dari (*SDSC) Blue Horizon logs* mengenai data tersebut (Steve Chapin, 2000).

9	430593	13	50866	256	25381	-1	256	64800	-1	1	95	-1	-1	4	-1	-1	-1
10	430730	31	10770	256	5343	-1	256	28800	-1	1	95	-1	-1	4	-1	-1	-1
11	433750	1878	12701	256	6307	-1	256	64500	-1	1	14	-1	-1	3	-1	-1	-1
12	437926	27	2794	64	2601	-1	64	60000	-1	1	34	-1	-1	2	-1	-1	-1
13	440070	22	43	32	0.03	-1	32	64800	-1	1	96	-1	-1	3	-1	-1	-1
14	440102	20	44	32	0.03	-1	32	64800	-1	1	96	-1	-1	3	-1	-1	-1
15	440357	14	43	32	0.03	-1	32	64800	-1	1	96	-1	-1	3	-1	-1	-1
16	440545	12	43	32	0.03	-1	32	64800	-1	1	96	-1	-1	3	-1	-1	-1
17	440599	19	43	32	0.03	-1	32	64800	-1	1	96	-1	-1	3	-1	-1	-1
18	440625	25	43	32	0.03	-1	32	64800	-1	1	96	-1	-1	3	-1	-1	-1
19	440999	0	84	32	0.03	-1	32	7200	-1	1	96	-1	-1	0	-1	-1	-1
20	441574	5	44	32	0.03	-1	32	64800	-1	1	96	-1	-1	3	-1	-1	-1
21	442667	31	7914	512	3903	-1	512	64500	-1	1	14	-1	-1	3	-1	-1	-1
22	443249	6	45	32	0.03	-1	32	64800	-1	1	96	-1	-1	3	-1	-1	-1
23	443330	0	84	8	-1	-1	8	7200	-1	1	96	-1	-1	0	-1	-1	-1
24	443363	0	50	8	-1	-1	8	7200	-1	1	96	-1	-1	0	-1	-1	-1
25	443797	4564	3260	128	1141	-1	128	14400	-1	1	68	-1	-1	4	-1	-1	-1

Gambar 4.3 Dataset SDSC Dari Versi *clean-4.2*

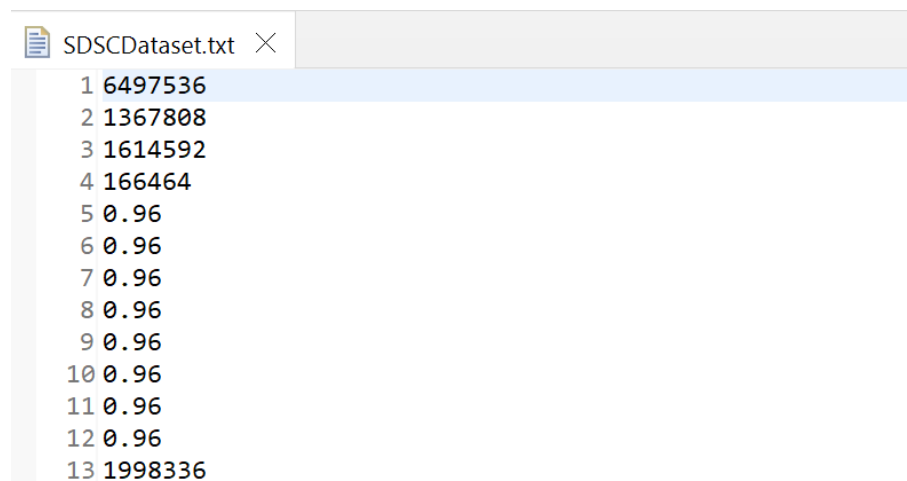
Penulis pun berusaha mencari jalan lain untuk bisa mendapatkan data tersebut dengan cara mengambil kolom ke-6 yaitu Total Waktu yang Dihabiskan di Prosesor dan kolom ke-8 yaitu Banyaknya Permintaan Prosesor. Dengan cara mengkalikan kedua nilai tersebut, penulis akan bisa mendapatkan besaran *Task* yang dimasukkan ke dalam sistem (*SDSC) Blue Horizon logs*. Penulis mengambil kedua nilai tersebut dan memasukkannya ke dalam *Microsoft Excel*. Namun, timbul masalah lain dimana kolom ke-6 dan kolom ke-8 tersebut pun juga memiliki nilai -1, sehingga penulis perlu menghapus nilai tersebut sehingga dari total 250.400 data yang dimiliki oleh dataset (*SDSC) Blue Horizon logs*, hanya 7.395 data yang bisa digunakan dalam penelitian ini. Hasil bersih dari *Preprocessing* dataset ini bisa dilihat pada Gambar 4.4.

	A	B	C	D
1	25381	256	6497536	
2	5343	256	1367808	
3	6307	256	1614592	
4	2601	64	166464	
5	0.03	32	0.96	
6	0.03	32	0.96	
7	0.03	32	0.96	
8	0.03	32	0.96	

Gambar 4.4 Hasil Bersih Dari *Preprocessing* Dataset

4.4.2 Membaca Nilai Dari Microsoft Excel

Sama seperti Bagian 4.3.2, karena Bahasa *Java* tidak bisa membaca nilai dari *Microsoft Excel* secara langsung, maka penulis memindahkan hasil pembuatan nilai acak dari Bagian 4.4.1 kepada *Notepad* yang penulis berikan nama *SDSCDataset.txt* seperti bisa dilihat pada Gambar 4.5 dibawah. Nantinya Bahasa *Java* akan bisa membaca nilai dari file *txt* tersebut.



Gambar 4.5 *SDSCDataset.txt*

Lalu sama seperti Bagian 4.3.2, penulis juga menggunakan metode *getSeedValue* untuk mengambil nilai dari *txt* tersebut. Cara kerja dari metode *getSeedValue* sudah dijelaskan pada bagian sebelumnya sehingga penulis tidak menjelaskan cara kerja metode tersebut lagi pada bagian ini.

4.4.3 Implementasi Pada Simulasi

Sama seperti Bagian 4.3.3, untuk bisa mengimplementasikan nilai dari dataset tersebut pada simulasi, Kode Sumber 4.1, yaitu kode untuk pengaturan *Task* akan melakukan pemanggilan pada metode *getSeedValue*. Hal ini akan berakibat, Kode Sumber 4.1 memiliki nilai dari tersebut dan bisa menambahkan nilai tersebut kepada panjang *Task* yang dibuat sehingga bisa terimplementasi dengan sempurna pada simulasi *Cloud*.

4.5 Implementasi Skenario Pertama

Untuk skenario pertama akan dilakukan implementasi menggunakan *Genetic Algorithm* saja. Implementasi skenario pertama ini akan dibagi menjadi beberapa sub-bab untuk mempermudah penjelasan. Sub-bab tersebut adalah pengaturan kromosom individu, pengambilan individu terbaik di dalam populasi, perhitungan *Fitness Function*, *Mutation*, *Crossover*, dan implementasi pada simulasi.

4.5.1 Pengaturan Kromosom Individu

Agar *Genetic Algorithm* bisa berjalan, maka penulis perlu melakukan pengaturan kromosom dari tiap individu yang ada di dalam populasi *Genetic Algorithm*. Kromosom dari tiap individu ini akan didasarkan pada kromosom sepanjang 9 yang dijelaskan pada Gambar 3.1. Setiap angka yang ada di dalam kromosom ini akan bernilai dari 0 hingga 8 dan merepresentasikan ID mesin virtual (VM) yang menjadi tempat alokasi *Task*.

```
this.chromosome = new int[chromosomeLength];
dataCenterIterator = dataCenterIterator-1;

int max = 8 + 9 * dataCenterIterator;
int min = 0 + 9 * dataCenterIterator;
int range = max - min + 1;

// generate random numbers within 0 to 8
for (int gene = 0; gene < chromosomeLength; gene++) {
    int rand = (int)(Math.random() * range) + min;
    this.setGene(gene, rand);
}
```

Kode Sumber 4.17 *Individual*

Kode Sumber 4.17 menunjukkan implementasi dari kromosom individu tersebut. Terlihat adanya variabel *dataCenterIterator* yang berguna untuk mengiterasikan penambahan angka 9 pada representasi ID mesin virtual (VM). Hal ini dikarenakan, simulasi *Cloud* memiliki 6 *Data Center* dimana *Data Center* ke-1 memiliki ID VM dari 0 hingga 8, *Data Center* ke-2 memiliki ID VM dari 9 hingga 17, *Data Center* ke-3 memiliki ID VM dari 18 hingga 26, dan seterusnya. Jika penulis tidak mengimplementasikan variabel *dataCenterIterator* tersebut, maka semua *Task* yang ada akan dijalankan pada *Data Center* ke-1 yang memiliki ID VM dari 0 hingga 8 saja.

Setelah penulis menambahkan jangkauan dari representasi ID mesin virtual (VM) tersebut, penulis langsung melakukan pengisian kromosom dengan 9 gen secara acak menggunakan fungsi *Math.random*. Hal ini akan berakibat individu di dalam *Genetic Algorithm* memiliki kromosom sepanjang 9 berisi angka acak dari jangkauan yang sesuai dengan urutan *Data Center* yang ada.

4.5.2 Pengambilan Individu Terbaik di Dalam Populasi

Pengambilan individu terbaik di dalam sebuah populasi dilakukan melalui sebuah algoritma Bubble Sort sederhana dimana setiap individu yang ada di dalam sebuah populasi akan diminta *Fitness Function*-nya dan akan diurutkan dari yang tertinggi hingga yang terkecil berdasarkan nilai tersebut. Setelah Sorting selesai dilakukan, akan didapatkan individu dengan

nilai Fitness Function tertinggi pada urutan pertama dan akan dijadikan keluaran dari metode. Metode ini dinamakan dengan nama `getFittest` dan implementasi kodenya bisa dilihat pada Kode Sumber 4.18 di bawah ini.

```
Arrays.sort(this.population, new Comparator<Individual>() {

    @Override
    public int compare(Individual o1, Individual o2) {
        if (o1.getFitness() < o2.getFitness()) {
            return 1;
        } else if (o1.getFitness() > o2.getFitness()) {
            return -1;
        }
        return 0;
    }
});

// Return the fittest individual
return this.population[offset];
```

Kode Sumber 4.18 `getFittest`

4.5.3 Perhitungan Fitness Function

Perhitungan *Fitness Function* terbagi menjadi 2 bagian berdasarkan Tabel 3.3 hingga Tabel 3.5 dimana terdapat 2 perhitungan yang harus penulis perhatikan yaitu Total Waktu Eksekusi dan Persentase Kegagalan. Perhitungan untuk Total Waktu Eksekusi membutuhkan panjang *Task* dan nilai *MIPS* dari tiap mesin virtual (VM) yang dituju oleh *Task* tersebut. Sedangkan Persentase Kegagalan membutuhkan *Failure Rate*, berapa banyak kesalahan yang akan terjadi berdasarkan Distribusi *Poisson*, dan berapa banyak kesalahan yang akan terjadi dalam sebuah Distribusi *Poisson Redundant* berjumlah 9 (Farouk A. Emara, 2021).

```
int iterator=0;
dataCenterIterator = dataCenterIterator-1;

for (int i=0 + dataCenterIterator*9 + cloudletIteration*54; i<9 +
dataCenterIterator*9 + cloudletIteration*54; i++)
{
    int gene = individual.getGene(iterator);
    if (gene%9 == 0 || gene%9 == 3 || gene%9 == 6)
    {
        mips = 400;
    }else if (gene%9 == 1 || gene%9 == 4 || gene%9 == 7)
    {
        mips = 500;
    }else if (gene%9 == 2 || gene%9 == 5 || gene%9 == 8)
    {
        mips = 600;
    }else break;

    totalExecutionTime = totalExecutionTime + cloudletList.get(i).
getCloudletLength()/ mips;
    iterator++;
}
```

Kode Sumber 4.19 Perhitungan *Fitness Function* Bagian Total Waktu Eksekusi

Untuk Total Waktu Eksekusi, implementasi kode ada pada Kode Sumber 4.19 diatas. Pertama-tama penulis membutuhkan ID dari *Task* yang akan dicari panjangnya. Cara mencarinya adalah dengan cara mengiterasikan variabel *dataCenterIterator* dikalikan dengan 9 karena tiap *Data Center* bisa menampung 9 *Task* ditambahkan dengan variabel *cloudletIteration* yang berguna untuk menyimpan pada iterasi siklus keberapakah *Task* sudah melewati 6 *Data Center* dan kembali ke *Data Center* pertama. Hasil penjumlahan ini akan memberikan penulis jangkauan ID *Task* untuk dicari panjangnya.

Lalu penulis ambil nilai gen dari tiap ID *Task* tersebut untuk mencari tahu berapakah *MIPS VM* yang akan mereka tuju. Lalu penulis ambil panjang dari tiap ID *Task* tersebut dan membagi nilai tersebut dengan *MIPS VM* yang akan mereka tuju. Hasil pembagian ini akan kita tambahkan secara *looping* hingga sudah 9 *Task* dicari dan nilai Total Waktu Eksekusi ditemukan.

Untuk Persentase Kegagalan, pertama-tama penulis perlu menemukan berapa banyak kesalahan yang akan terjadi berdasarkan Distribusi *Poisson* (Farouk A. Emara, 2021). Menurut algoritma yang diberikan oleh Donald Knuth, seorang Programmer dari Amerika Selatan, Distribusi *Poisson* bisa diberikan dalam Kode Sumber 4.20 di bawah ini (Knuth, 1968). Sehingga ketika diubah menjadi Bahasa *Java* akan menjadi Kode Sumber 4.21

```

algorithm poisson random number (Knuth):
  init:
    Let  $L \leftarrow e^{-\lambda}$ ,  $k \leftarrow 0$  and  $p \leftarrow 1$ .
  do:
     $k \leftarrow k + 1$ .
    Generate uniform random number  $u$  in  $[0,1]$  and let  $p \leftarrow p \times u$ .
  while  $p > L$ .
  return  $k - 1$ .

```

Kode Sumber 4.20 *Pseudocode* Distribusi *Poisson*

```

public static int getRandomPoisson(double lambda)
{
    double L = Math.exp(-lambda);
    double p = 1.0;
    int k = 0;

    do
    {
        k++;
        p *= Math.random();
    } while (p > L);

    return k - 1;
}

```

Kode Sumber 4.21 Distribusi *Poisson* Dalam Bahasa *Java*

Dalam Kode Sumber 4.21, diminta 1 parameter yaitu nilai *Lambda* dimana nilai ini berarti adalah *Failure Rate* atau kemungkinan sebuah *Task* gagal untuk dilaksanakan di dalam simulasi *Cloud*. Penulis mengambil nilai ini dari hasil simulasi milik *The San Diego Supercomputer Center (SDSC) Blue Horizon logs* (log, 2003). Dalam simulasi mereka ditemukan bahwa nilai *Failure Rate* adalah 0.04847. Nilai ini penulis masukkan di dalam Distribusi *Poisson* tersebut untuk mensimulasikan terjadinya kegagalan pelaksanaan *Task* di dalam simulasi.

Tetapi masih ada yang perlu penulis lakukan karena Kode Sumber 4.21 hanya bisa digunakan untuk mensimulasikan kesalahan hanya pada satu saja mesin virtual (VM) sedangkan ada 9 mesin virtual (VM) di dalam setiap representasi kromosom di *Genetic Algorithm*. Hal ini bisa diselesaikan dengan menggunakan Distribusi *Poisson Redundant*, yang mana penulis mengiterasikan hasil nilai dari Distribusi *Poisson* pada Kode Sumber 4.21 dengan kemungkinan *Failure Rate* sebanyak 9 mesin virtual (VM). Implementasi kode Distribusi *Poisson Redundant* ini bisa dilihat pada Kode Sumber 4.22 dibawah ini dimana nilai variabel *x* adalah hasil dari Distribusi *Poisson* pada Kode Sumber 4.21 dan nilai variabel *n* adalah banyak mesin virtual (VM) yaitu 9.

```
public static double getPoisson(double lambda, int x, int n)
{
    double L = Math.exp(-lambda) * n;
    double p = Math.pow(lambda * n, x);
    int k = factorial(x);
    double result;

    result = L * p / k;

    return result;
}
```

Kode Sumber 4.22 Distribusi *Poisson Redundant* Dalam Bahasa Java

Setelah kedua bagian dari perhitungan *Fitness Function* selesai dibuat, maka penulis melakukan pemanggilan kedua metode tersebut untuk mendapatkan Total Waktu Eksekusi dan Persentase Kegagalan dan mengkalikan mereka berdua dengan persentase. Persentase ini menunjukkan seberapa penting kedua bagian ini dimana penulis memberikan nilai 90% untuk Total Waktu Eksekusi dan 10% untuk Persentase Kegagalan karena penelitian ini lebih menitik-beratkan kepada meningkatkan penggunaan sumber daya *Cloud* daripada mencegah terjadinya kesalahan dalam implementasi *Task*. Implementasi bagian terakhir dari *Fitness Function* bisa dilihat pada Kode Sumber 4.23 di bawah ini.

```
int random = getRandomPoisson(failureRate);
double poisson=(getPoisson(failureRate, random, 9));

// Calculate fitness
double fitness = 0.90 * (1/totalExecutionTime) + 0.1 * (1/poisson);

// Store fitness
individual.setFitness(fitness);
return fitness;
```

Kode Sumber 4.23 Bagian Terakhir *Fitness Function*

4.5.4 Mutation

Mutation adalah perubahan secara acak di dalam sebuah nilai gen di dalam representasi kromosom *Genetic Algorithm*. *Mutation* cukup penting dalam memberikan variasi terhadap gen yang ada di dalam sebuah populasi. Tanpa adanya *Mutation*, pencarian individu yang memiliki nilai *Fitness Function* yang tinggi akan menjadi sangat lama untuk ditemukan. Tetapi jika terlalu banyak terjadi *Mutation*, maka individu di dalam populasi akan memiliki gen yang terlalu acak sehingga tidak bisa didapatkan kenaikan yang signifikan terhitung dari *Fitness Function*.

Implementasi kode untuk *Mutation* bisa dilihat pada Kode Sumber 4.24 di bawah. Untuk bisa mengimplementasikan *Mutation*, penulis mengeluarkan nilai acak menggunakan fungsi *Math.random*. Apabila nilai yang dihasilkan lebih rendah daripada *Mutation Rate* yang telah di definisikan pada Tabel 3.2, maka *Mutation* dilakukan. Lalu penulis hanya perlu mengganti 1 gen acak dengan 1 nilai gen yang lain secara acak dan terjadilah mutasi di dalam kromosom individu tersebut.

```
// Does this gene need mutation?  
if (this.mutationRate > Math.random()) {  
    // Get new gene  
    int newGene=0 + 9 * dataCenterIterator;  
    double r=Math.random() * range;  
    newGene=r + 9 * dataCenterIterator;  
}  
// Mutate gene  
individual.setGene(geneIndex, newGene);
```

Kode Sumber 4.24 Potongan Kode *Mutation*

4.5.5 Crossover

Crossover adalah penyilangan gen dari 2 orangtua untuk menghasilkan 1 individu anak yang baru di generasi berikutnya. Pemilihan 2 orangtua didasarkan pada 1 orangtua yang memiliki nilai *Fitness Function* yang tinggi dan orangtua yang lain akan dipilih secara acak. Gen akan diambil secara acak dari orangtua pertama dan orangtua kedua hingga didapatkan 9 gen untuk bisa menghasilkan individu baru. Proses *Crossover* adalah proses paling penting di dalam *Genetic Algorithm* karena merupakan inti dari cara kerjanya untuk menghasilkan generasi yang baru dengan peningkatan nilai *Fitness Function*.

Implementasi kode untuk *Crossover* bisa dilihat pada Kode Sumber 4.25 di bawah. Untuk bisa mengimplementasikan *Crossover*, pertama-tama penulis mengambil orangtua pertama dengan nilai *Fitness Function* yang tinggi, lalu mengambil orangtua kedua secara acak di dalam populasi yang bukan orangtua pertama. Lalu penulis mengeluarkan nilai acak menggunakan fungsi *Math.random*. Jika nilainya dibawah 0.5 maka, penulis mengambil gen pertama milik orangtua pertama dan sebaliknya jika nilainya diatas 0.5 maka, penulis mengambil gen pertama milik orangtua kedua. Proses ini akan dilanjutkan terus hingga telah didapatkan 9 gen untuk individu baru. Dan dengan demikian proses *Crossover* sudah berhasil diimplementasikan di dalam simulasi *Cloud*.

Proses *Crossover* ini akan terus dilakukan hingga didapatkan satu generasi baru secara penuh. Jika belum didapatkan satu generasi baru secara penuh, maka proses *Crossover* akan

dijalankan terus menerus hingga semua individu didapatkan dan di-Assign kepada generasi yang baru.

```
// Create new population
Population newPopulation = new Population(population.size());

// Loop over current population by fitness
for (int populationIndex = 0; populationIndex < population.size(); populationIndex++)
{
    Individual parent1 = population.getFittest(populationIndex);

    // Apply crossover to this individual?
    if (this.crossoverRate > Math.random() && populationIndex > this.elitismCount ) {
        // Initialize offspring
        Individual offspring = new Individual(parent1.getChromosomeLength(),
        dataCenterIterator);

        // Find second parent
        Individual parent2 = selectParent(population);

        // Loop over genome
        for (int geneIndex = 0; geneIndex < parent1.getChromosomeLength(); geneIndex++) {
            // Use half of parent1's genes and half of parent2's genes
            if (0.5 > Math.random()) {
                offspring.setGene(geneIndex, parent1.getGene(geneIndex));
            } else {
                offspring.setGene(geneIndex, parent2.getGene(geneIndex));
            }
        }

        // Add offspring to new population
        newPopulation.setIndividual(populationIndex, offspring);
    } else {
        // Add individual to new population without applying crossover
        newPopulation.setIndividual(populationIndex, parent1);
    }
}
```

Kode Sumber 4.25 Potongan Kode *Crossover*

4.5.6 Implementasi Pada Simulasi

Setelah semua potongan bagian dari *Genetic Algorithm* diimplementasikan, sekarang penulis harus mengimplementasikan *Genetic Algorithm* kedalam simulasi *Cloud* di dalam *CloudSim*. Untuk bisa melakukan hal tersebut, penulis perlu mendefenisikan 2 variabel untuk memastikan tiap *Task* dan tiap mesin virtual (VM) yang ada di dalam simulasi bisa terbaca dan diikuti sertakan dalam simulasi. Variabel pertama adalah *dataCenterIterator* yang berfungsi untuk mengiterasikan tiap *Data Center* sehingga ke semua 54 mesin virtual (VM) yang ada bisa terbaca. Variabel kedua adalah *cloudletIterator* yang berfungsi untuk mengiterasikan setiap *Task* yang ada karena ID VM akan mengulang tiap 54 tetapi ID *Task* tidak akan mengulang dan berlanjut hingga habis. Untuk hal ini penulis mengimplementasikan kedua variabel tersebut. Hasil implementasi kode bisa dilihat pada Kode Sumber 4.26 dan dengan demikian *Genetic Algorithm* sudah bisa terimplementasi dengan baik.

```

for (int cloudletIterator=0; cloudletIterator<=cloudletLoopingNumber;
cloudletIterator++)
{
for (int dataCenterIterator = 1; dataCenterIterator <= 6; dataCenterIterator++)
{
    // Initialize Genetic Algorithm
    GeneticAlgorithm ga = new GeneticAlgorithm(20, 0.3, 0.95, 2, cloudletList,
vmlist);

    // Initialize population
    Population population = ga.initPopulation(chromosomeLength,
dataCenterIterator);

    // Evaluate population
    ga.evalPopulation(population, dataCenterIterator, cloudletIterator);

    // Genetic Algorithm Iteration
    int iteration = 1;
    while (iteration <= 15)
    {
        // get fittest individual from population in every iteration
        Individual fit = population.getFittest(0);
        // Apply crossover
        population = ga.crossoverPopulation(population, dataCenterIterator);
        // Apply mutation
        population = ga.mutatePopulation(population, dataCenterIterator);
        // Evaluate population
        ga.evalPopulation(population, dataCenterIterator, cloudletIterator);

        // Increment the current generation
        iteration++;
    }

    for (int assigner=0+(dataCenterIterator-1)*9 + cloudletIterator*54;
assigner<9+(dataCenterIterator-1)*9 + cloudletIterator*54; assigner++)
    {
        broker.bindCloudletToVm(assigner,
population.getFittest(0).getGene(assigner%9));
    }
}
}

```

Kode Sumber 4.26 Potongan Kode Simulasi *Genetic Algorithm*

4.6 Implementasi Skenario Kedua

Untuk skenario kedua akan dilakukan implementasi menggunakan *Genetic Algorithm* dan *Artificial Neural Network*. Implementasi skenario pertama ini akan dibagi menjadi beberapa sub-bab untuk mempermudah penjelasan. Sub-bab tersebut adalah *Preprocessing* pada dataset, membaca nilai dari *Microsoft Excel*, pembuatan struktur *Artificial Neural Network*, normalisasi data, proses pembelajaran data, proses pengujian data, dan implementasi pada simulasi.

4.6.1 Preprocessing Pada Dataset

Setelah skenario pertama selesai dijalankan menggunakan *Genetic Algorithm*, penulis bisa mendapatkan dataset berupa ID *Task*, panjang *Task*, ID mesin virtual (VM), *MIPS* mesin virtual (VM), dan juga jadwal yang diberikan sebagai keluaran dari penggunaan *Genetic Algorithm*. Dataset ini bisa dilihat pada Gambar 4.6 dibawah ini.

A	B	C	D	E
ID Task	Panjang Task	ID VM	MIPS VM	Jadwal
{0,1,2,3,4,5,6,7,8}	{24417,15991,25064,23516,47939,40183,29131,20415,40787}	{0,1,2,3,4,5,6,7,8}	{400,500,600,400,500,600,400,500,600}	{5,5,5,2,8,2,8,5,5}
{9,10,11,12,13,14,15,16,17}	{46048,39321,24989,40660,30501,24252,44641,49752,11026}	{9,10,11,12,13,14,15,16,17}	{400,500,600,400,500,600,400,500,600}	{17,10,17,12,16,16,16,9,13}
{18,19,20,21,22,23,24,25,26}	{48186,16379,24756,18759,24901,27014,34570,26119,15275}	{18,19,20,21,22,23,24,25,26}	{400,500,600,400,500,600,400,500,600}	{20,26,26,26,22,20,23,25,20}
{27,28,29,30,31,32,33,34,35}	{20442,35568,32794,29395,42320,50543,37309,14345,46117}	{27,28,29,30,31,32,33,34,35}	{400,500,600,400,500,600,400,500,600}	{32,35,32,29,35,35,28,27,31}
{36,37,38,39,40,41,42,43,44}	{25801,18557,39608,27964,39700,42033,33285,36571,47429}	{36,37,38,39,40,41,42,43,44}	{400,500,600,400,500,600,400,500,600}	{43,43,38,44,44,43,44,43,38}
{45,46,47,48,49,50,51,52,53}	{33067,13054,15829,24408,29147,15743,45882,42888,34150}	{45,46,47,48,49,50,51,52,53}	{400,500,600,400,500,600,400,500,600}	{53,50,53,53,53,48,47,50,53}
{54,55,56,57,58,59,60,61,62}	{43728,34015,44203,38145,50450,47851,37459,34319,34913}	{0,1,2,3,4,5,6,7,8}	{400,500,600,400,500,600,400,500,600}	{8,2,5,2,4,5,8,2,7}

Gambar 4.6 Dataset Awal Hasil Keluaran *Genetic Algorithm*

Pada awalnya penulis menggabungkan kedua kelas *Task* menjadi 1 *array*-2-dimensi dan menggabungkan kedua kelas mesin virtual (VM) juga menjadi 1 *array*-2-dimensi. *Input Node* dari *Artificial Neural Network* akan menerima kedua *array*-2-dimensi ini sebagai masukkannya. Sebagai keluarannya, *Artificial Neural Network* akan mengeluarkan 1 *array*-1-dimensi di *Output Node*-nya.

Namun di *Java*, hal ini tidak memungkinkan untuk dilakukan karena secara teori awal yang sudah dijelaskan di Bagian 2.2.7, *Artificial Neural Network* tidak bisa menerima *Array* sebagai masukkan dan keluarannya, apalagi *Array-Multi-dimensional*. Dengan demikian, penulis melakukan *Preprocessing* pada dataset sehingga hanya tersisa kelas panjang *Task* dan kelas jadwal saja. Kelas panjang *Task* akan menjadi masukkan yang diterima oleh 9 *Input Node* dan kelas jadwal akan menjadi kelas keluaran yang akan diberikan oleh 9 *Output Node*. Di antara kedua *Node* tersebut akan ada 1 *Hidden Node* berjumlah 18 yang akan membantu *Artificial Neural Network* dalam memprediksi jadwal sebagai keluarannya (Heaton, Programming Neural Networks with Encog3 in Java, 2011).

Ada 1 hal lagi yang perlu diperhatikan yaitu, pada simulasi *Cloud* menggunakan *Genetic Algorithm*, ID dari mesin virtual (VM) memiliki jangkauan dari angka 0 hingga angka 53. Hal ini tidak diinginkan oleh penulis karena seharusnya *Artificial Neural Network* hanya diperlukan untuk menebak pada mesin virtual (VM) mana sebuah *Task* dengan panjang sekian dialokasikan. *Artificial Neural Network* tidak diharuskan untuk menebak pada *Data Center* berapa *Task* tersebut dialokasikan, oleh karena itu dataset jadwal harus di modulo dengan 9 terlebih dahulu. Setelah semua proses *Preprocessing* ini selesai maka didapatkanlah dataset yang bersih seperti bisa dilihat pada Gambar 4.7 dibawah ini.

Panjang Task	Jadwal
{24417,15991,25064,23516,47939,40183,29131,20415,40787}	{3,8,7,2,5,1,6,5,2}
{46048,39321,24989,40660,30501,24252,44641,49752,11026}	{2,2,2,2,7,4,8,2,6}
{48186,16379,24756,18759,24901,27014,34570,26119,15275}	{2,8,8,4,8,8,2,8,7}
{20442,35568,32794,29395,42320,50543,37309,14345,46117}	{5,2,5,5,2,8,1,4,8}
{25801,18557,39608,27964,39700,42033,33285,36571,47429}	{0,8,4,3,2,2,4,3,7}
{33067,13054,15829,24408,29147,15743,45882,42888,34150}	{2,8,3,4,8,6,2,8,5}
{43728,34015,44203,38145,50450,47851,37459,34319,34913}	{2,1,1,8,6,5,7,3,1}

Gambar 4.7 Dataset Akhir Setelah *Preprocessing*

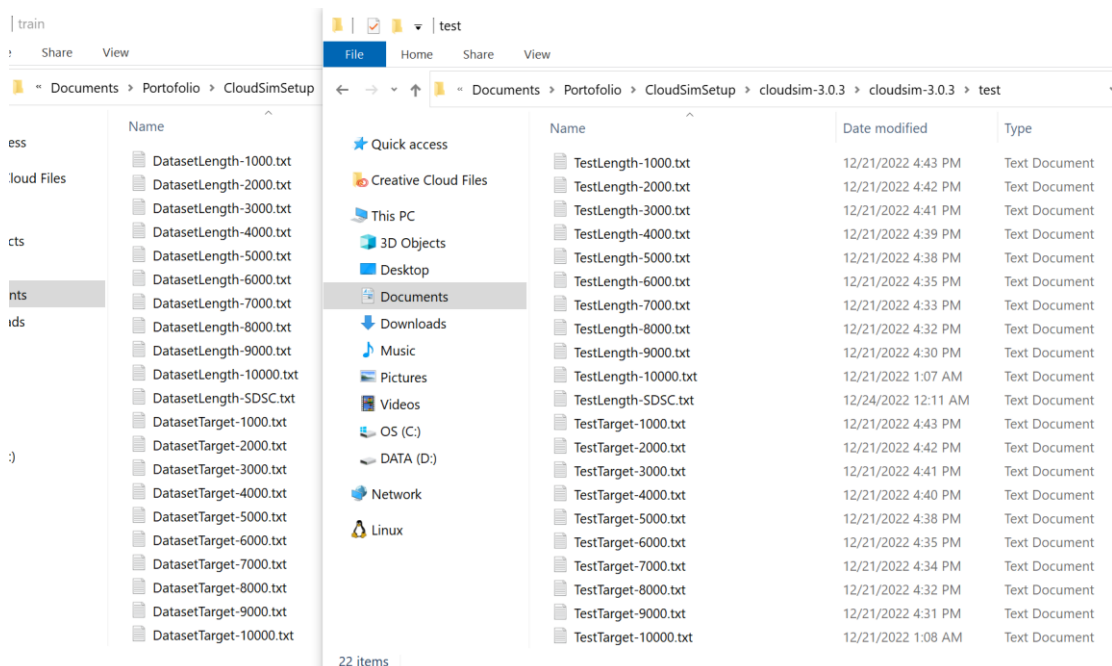
4.6.2 Membaca Nilai Dari Microsoft Excel

Sama seperti pada bagian-bagian sebelumnya, karena Bahasa *Java* tidak bisa membaca nilai dari *Microsoft Excel* secara langsung, maka penulis memindahkan hasil pembuatan dataset dari Bagian 4.6.1 kepada *Notepad* agar *Java* bisa membaca dari file *txt*. Karena dataset perlu dibagi menjadi 80% dataset pembelajaran dan 20% dataset pengujian, maka penulis melakukan pembagian berdasarkan Tabel 4.1 dibawah ini.

Tabel 4.1 Pembagian Dataset

Nama Dataset	Jumlah Data Awal	Data Pembelajaran	Data Pengujian
Acak	111	89	22
Acak	222	178	44
Acak	333	267	66
Acak	444	356	88
Acak	555	444	111
Acak	666	533	133
Acak	777	622	155
Acak	888	711	177
Acak	1000	800	200
Acak	1110	888	222
SDSC	816	653	163

Penulis juga membagi kelas panjang *Task* sebagai masukkan dan kelas jadwal sebagai keluaran ke 2 file *txt* yang berbeda untuk mempermudah pembacaan dari *Java*. Kelas panjang *Task* memiliki nama *Length* pada filenya dan kelas jadwal memiliki nama *Target* pada filenya Hasil akhir dari pengubahan data dari *Microsoft Excel* ke file *txt* bisa dilihat pada Gambar 4.8 dibawah ini.



Gambar 4.8 Hasil Akhir Setelah Pembagian Dataset

Setelah selesai mengubah data dari *Microsoft Excel* ke file *txt*, penulis harus bisa membaca isi dari file tersebut menjadi masukkan ke dalam simulasi. Untuk bisa melakukannya, penulis mengimplementasikan 2 metode yaitu, *Reading2DArrayFromFileLength* dan *Reading2DArrayFromFileTarget*. Kedua metode ini pada prinsipnya sama hanya saja berbeda di file yang mereka baca dimana metode *Reading2DArrayFromFileLength* membaca kelas panjang *Task* dan metode *Reading2DArrayFromFileTarget* akan membaca kelas jadwal. Implementasi kode bisa dilihat pada Kode Sumber 4.27 dibawah ini.

```
Scanner scannerLength;
int rows = 653; // Number of rows to be scanned
int columns = 9; // Number of columns to be scanned
double [][] arrayLength = new double[rows][columns];

try
{
    scannerLength = new Scanner(new BufferedReader(new
    FileReader(System.getProperty("user.dir")+ "/train/DatasetLength-SDSC.txt")));
    while(scannerLength.hasNextLine()) {
        for (int i=0; i<arrayLength.length; i++) {
            String[] line = scannerLength.nextLine().trim().split(" ");
            for (int j=0; j<line.length; j++) {
                arrayLength[i][j] = Integer.parseInt(line[j]); // Parsing String to Integer
                and save to array
            }
        }
    }
} catch (FileNotFoundException e)
{
    e.printStackTrace();
}
return arrayLength;
```

Kode Sumber 4.27 Metode Membaca Dari File Txt

4.6.3 Pembuatan Struktur Artificial Neural Network

Pembuatan struktur dari *Artificial Neural Network* akan menggunakan bantuan *Library Encog* yang didesain khusus untuk pembuatan *Artificial Neural Network*. Disini penulis hanya perlu mendefinisikan nama dari *Artificial Neural Network* dan menambahkan *Layer* beserta *Activation Function* berdasarkan spesifikasi dari Tabel 3.6 dan Tabel 3.7. Implementasi kode bisa dilihat pada Kode Sumber 4.28 dibawah ini.

```
BasicNetwork network = new BasicNetwork();
network.addLayer(new BasicLayer(null,true,9)); // 9 input nodes
network.addLayer(new BasicLayer(new ActivationReLU(),true,18)); // 18 hidden nodes
network.addLayer(new BasicLayer(new ActivationSigmoid(),false,9)); // 9 output nodes
network.getStructure().finalizeStructure();
network.reset();
```

Kode Sumber 4.28 Pembuatan Struktur *Artificial Neural Network*

4.6.4 Normalisasi Data

Sebelum *Artificial Neural Network* bisa memprediksi data dari dataset yang disediakan, normalisasi data diperlukan. Normalisasi diperlukan agar rentang dari data masukkan dan data keluaran tetap konsisten. Pada penelitian ini, kelas panjang *Task* dan kelas jadwal memiliki rentang data yang berbeda dimana panjang *Task* memiliki rentang dari 10000 hingga 40000 untuk dataset acak dan rentang dari 0 hingga 8790000. Sedangkan kelas jadwal hanya memiliki rentang dari 0 hingga 10. Disinilah terjadinya perbedaan konsistensi rentang data sehingga diperlukan adanya normalisasi data.

Normalisasi data akan menggunakan bantuan *Library Encog* dimana sudah disediakan kelas untuk normalisasi sehingga penulis hanya perlu melakukan pemanggilan metode *NormalizedField* untuk memberikan peraturan normalisasi seperti rentang data awal dan rentang data yang diinginkan. Lalu penulis memasukkan data dari Bagian 4.6.2 kepada aturan normalisasi data tersebut dan data tersebut akan ternormalisasi secara otomatis. Implementasi kode ini bisa dilihat pada Kode Sumber 4.29 di bawah ini.

```
// Saving the data scanned into the double arrays
LENGTH_RAW_DATA = Reading2DArrayFromFileLength();
TARGET_RAW_DATA = Reading2DArrayFromFileTarget();

//NormalizedField input = new NormalizedField(NormalizationAction.Normalize, null,
50000, 10000, 1, 0); //for Random Dataset
NormalizedField input = new NormalizedField(NormalizationAction.Normalize, null,
8790000, 0, 1, 0); //for SDSC
NormalizedField output = new NormalizedField(NormalizationAction.Normalize, null, 10,
0, 1, 0);

// Doing normalization to the Input
for (int m=0; m<LENGTH_RAW_DATA.length; m++) {
    for (int n=0; n<9; n++) {
        LENGTH_RAW_DATA[m][n] = input.normalize(LENGTH_RAW_DATA[m][n]);
    }
}

// Doing normalization to the Output
for (int m=0; m<TARGET_RAW_DATA.length; m++) {
    for (int n=0; n<9; n++) {
        TARGET_RAW_DATA[m][n] = output.normalize(TARGET_RAW_DATA[m][n]);
    }
}
```

Kode Sumber 4.29 Normalisasi Data

4.6.5 Proses Pembelajaran Data

Setelah normalisasi data selesai dilakukan, sekarang penulis melakukan proses pembelajaran data dimana *Artificial Neural Network* mulai belajar untuk melakukan prediksi data berdasarkan dataset yang diberikan. Proses pembelajaran ini akan dilakukan dengan cara membuat *TrainingSet* yang terdiri dari 2 parameter yaitu kelas masukkan yang mana adalah panjang *Task* dan kelas keluaran yang mana adalah jadwal.

Setelah *TrainingSet* selesai dibuat penulis melakukan pelatihan *Artificial Neural Network* menggunakan propagasi *Manhattan* hingga *Epoch* mencapai 100.000 atau akurasi mencapai 88% sesuai dengan spesifikasi *Artificial Neural Network* pada Tabel 3.6. Implementasi kode pelatihan bisa dilihat pada Kode Sumber 4.30 dibawah ini.

```
// Create training data
MLDataSet trainingSet = new BasicMLDataSet(LENGTH_RAW_DATA, TARGET_RAW_DATA);

// Train the neural network
final ManhattanPropagation train = new ManhattanPropagation(network, trainingSet,
0.00001);
int epoch = 1;

do {
    train.iteration();
    epoch++;
} while(epoch<100000 && train.getError()>0.12);
train.finishTraining();
```

Kode Sumber 4.30 Proses Pembelajaran *Artificial Neural Network*

4.6.6 Proses Pengujian Data

Setelah proses pembelajaran data selesai dilakukan, sekarang saatnya dilakukan proses pengujian data. Untuk bisa melakukan proses ini, penulis menggunakan dataset yang sama dengan dataset yang digunakan pada proses pembelajaran data. Hal ini dilakukan untuk memastikan data hasil prediksi dari *Artificial Neural Network* memberikan keluaran yang diinginkan oleh penulis. Nantinya setelah melalui proses ini, barulah *Artificial Neural Network* akan menggunakan dataset pengujian.

Proses pelatihan ini cukup sederhana dimana penulis hanya perlu melakukan pemanggilan metode *compute* untuk menjalankan *Artificial Neural Network* yang sudah terlatih pada dataset yang diinginkan. Lalu penulis melakukan denormalisasi untuk merubah data yang sudah dinormalisasi menjadi seperti semula. Terakhir, penulis melakukan penyimpanan struktur *Artificial Neural Network* tersebut untuk bisa digunakan pada simulasi sesungguhnya. Hasil implementasi kode bisa dilihat pada Kode Sumber 4.31 dibawah ini.

```
// Test the neural network
System.out.println("Neural Network Results:");
for(MLDataPair pair: trainingSet ) {
    final MLData outputData = network.compute(pair.getInput());
    for (int a=0 ; a<9; a++) {
        System.out.print(Math.round(input.deNormalize(pair.getInput().getData(a))));
    }
    for (int b=0 ; b<9; b++) {
        System.out.print(Math.round(output.deNormalize(outputData.getData(b))));
    }
    for (int c=0 ; c<9; c++) {
        System.out.print(Math.round(output.deNormalize(pair.getIdeal().getData(c))));
    }
}

// Saving the neural network
EncogDirectoryPersistence.saveObject(new File("ANNscheduler.EG"), network);
Encog.getInstance().shutdown();
```

Kode Sumber 4.31 Proses Pengujian *Artificial Neural Network*

4.6.7 Implementasi Pada Simulasi

Setelah semua potongan bagian dari *Artificial Neural Network* diimplementasikan, sekarang penulis harus mengimplementasikan *Artificial Neural Network* kedalam simulasi *Cloud* di dalam *CloudSim*. Untuk bisa melakukan hal tersebut, penulis harus terlebih dahulu membuang 80% dari kedua dataset yaitu *RandomDataset.txt* dan *SDSCDataset.txt* sehingga menyisakan 20% yang akhir untuk digunakan dalam simulasi. Pembagian ini dilakukan berdasarkan Tabel 4.2 dibawah.

Tabel 4.2 Pembagian Dataset Panjang *Task*

Nama Dataset	Jumlah Data Awal	Data Pengujian
<i>RandomDataset.txt</i>	1000	200
<i>RandomDataset.txt</i>	2000	400
<i>RandomDataset.txt</i>	3000	600
<i>RandomDataset.txt</i>	4000	800
<i>RandomDataset.txt</i>	5000	1000
<i>RandomDataset.txt</i>	6000	1200
<i>RandomDataset.txt</i>	7000	1400
<i>RandomDataset.txt</i>	8000	1600
<i>RandomDataset.txt</i>	9000	1800
<i>RandomDataset.txt</i>	10000	2000
<i>SDSCDataset.txt</i>	7395	1479

Setelah melakukan pembagian dataset pada Tabel 4.2, penulis bisa membaca nilai dari dataset tersebut menggunakan metode *getSeedValue* pada Bagian 4.3.2. Lalu penulis membaca dataset pengujian yang sudah dibagi pada Bagian 4.6.2 menggunakan metode *Reading2DArrayFromFileLength* dan *Reading2DArrayFromFileTarget*. Setelah melakukan pembacaan data, penulis melakukan normalisasi data seperti pada Bagian 4.6.4 untuk menyamakan rentang dataset. Metode-metode ini sudah dijelaskan di bagian-bagian sebelumnya dan tidak akan dijelaskan kembali pada bagian ini.

Setelah semua data siap, maka penulis melakukan *Load* terhadap *Artificial Neural Network* yang sudah disimpan dari Bagian 4.6.6 dan memasukkan dataset yang sudah terbaca sebagai masukan kepada *Artificial Neural Network*. *Artificial Neural Network* kemudian akan membaca masukan dan memberikan prediksi keluaran berupa jadwal mesin virtual (VM) yang harus dituju oleh masing-masing *Task*.

Namun ada masalah karena jadwal yang dikeluarkan hanya memiliki jangkauan 0 hingga 8 yang mana adalah ID mesin virtual di *Data Center* pertama saja. Untuk bisa menyelesaikan permasalahan ini, penulis memberikan variabel *iterator* yang akan mengiterasikan penambahan angka 9 kepada hasil keluaran dari *Artificial Neural Network* sehingga jadwal yang dikeluarkan memiliki jangkauan 6 *Data Center* yang tersedia di dalam lingkungan simulasi *Cloud*. Hasil implementasi kode bisa dilihat pada Kode Sumber 4.32 dibawah dan dengan demikian skenario kedua, yaitu implementasi algoritma *Genetic Algorithm* bersamaan dengan *Artificial Neural Network* berhasil diimplementasikan.

```

BasicNetwork network = (BasicNetwork)EncogDirectoryPersistence.loadObject(new
File("ANNscheduler-SDSC.eg"));

// Create data
MLDataSet trainingSet = new BasicMLDataSet(LENGTH_RAW_DATA, TARGET_RAW_DATA);
int iterator = 0; //Iterator for the Cloudlet IDs
Long placeholderLong; //Placeholder to convert long to integer

// Testing the ANN
for(MLDataPair pair: trainingSet ) {
    final MLData outputData = network.compute(pair.getInput());
    for (int a=0 ; a<9; a++) {
        System.out.print(Math.round(input.deNormalize(pair.getInput().getData(a))));
    }
    for (int b=0 ; b<9; b++) {
        System.out.print(Math.round(output.deNormalize(outputData.getData(b))));
    }
    for (int c=0 ; c<9; c++) {
        placeholderLong = new Long(Math.round(output.deNormalize(outputData.getData(c))));
        int VMidOutput = placeholderLong.intValue();
        System.out.print(cloudletList.get(iterator*9+c).getCloudletId());
        broker.bindCloudletToVm(cloudletList.get(iterator*9+c).getCloudletId(),
(VMidOutput + iterator*9)%54);
    }
    iterator++;
}

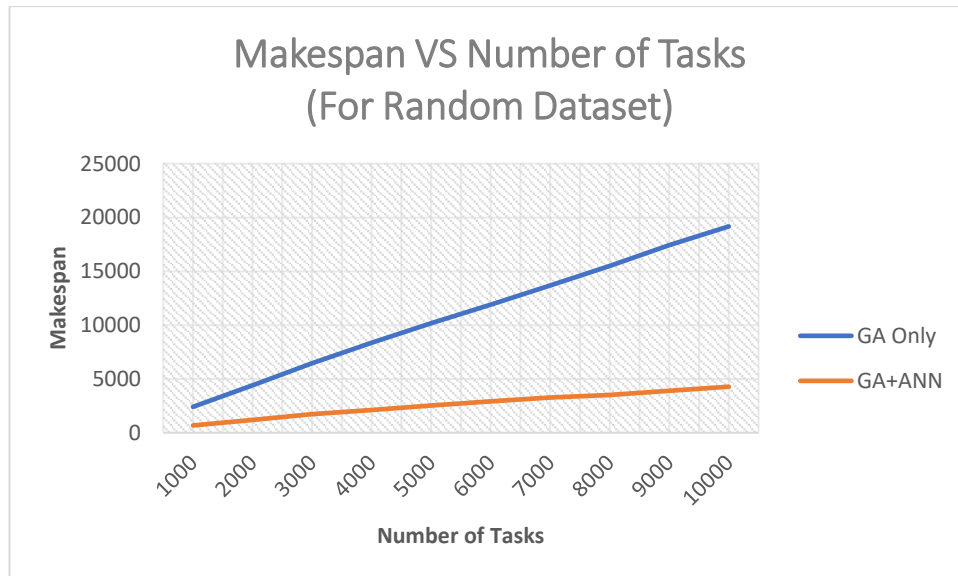
```

Kode Sumber 4.32 Potongan Kode Simulasi *Artificial Neural Network*

BAB V

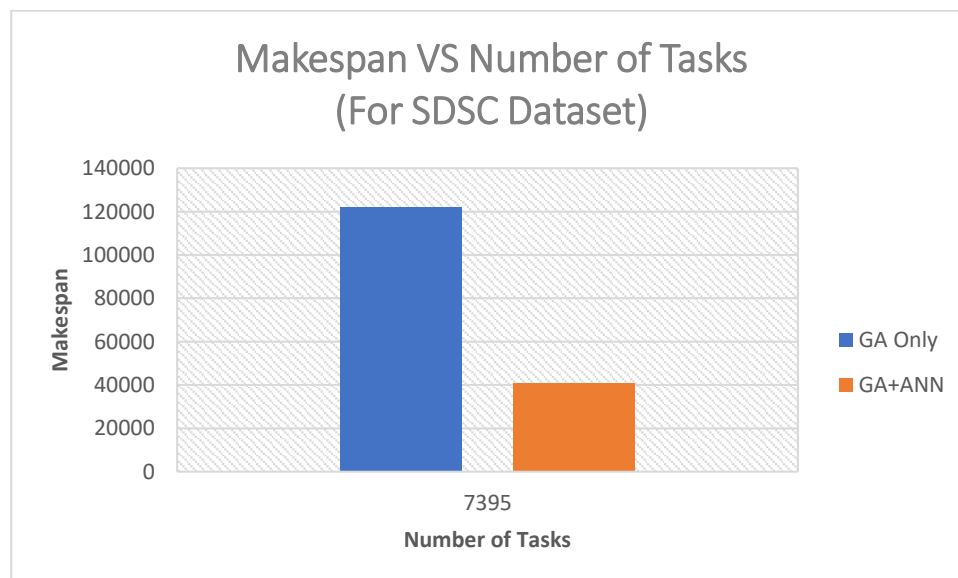
UJI COBA DAN ANALISIS

5.1 Makespan



Gambar 5.1 Grafik Perbandingan *Makespan* Pada Dataset Acak

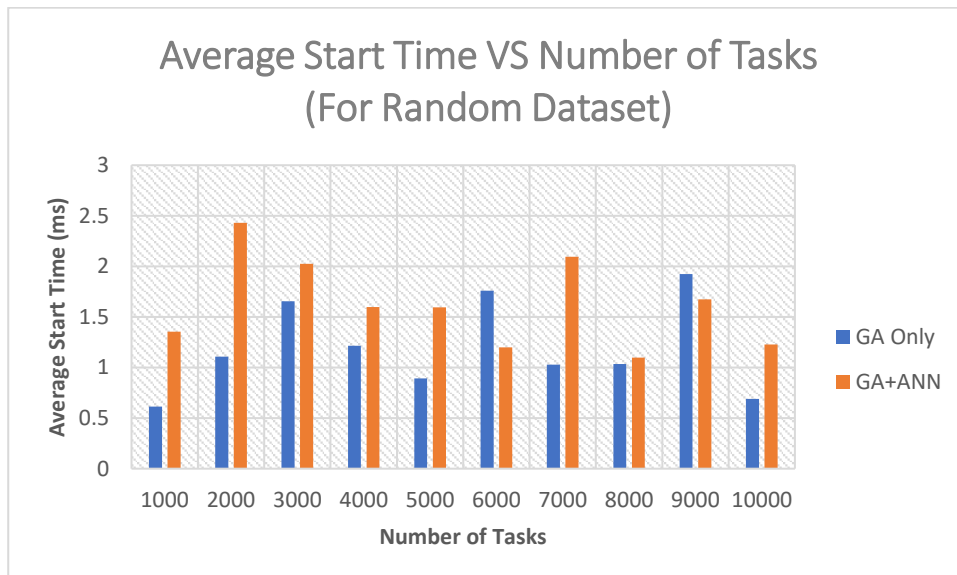
Makespan berarti waktu *Task* terakhir berhasil diselesaikan dan semakin rendah nilainya maka semakin baik sebuah algoritma. Dari Gambar 5.1 bisa dilihat bahwa skenario kedua, yaitu implementasi *Genetic Algorithm* bersamaan dengan *Artificial Neural Network* memiliki nilai yang lebih rendah secara konsisten dengan adanya penambahan jumlah *Task*. Hal ini berarti *Task* terakhir yang ada di dalam simulasi skenario kedua lebih cepat diselesaikan oleh implementasi *Genetic Algorithm* bersamaan dengan *Artificial Neural Network* dibandingkan dengan implementasi *Genetic Algorithm* saja.



Gambar 5.2 Grafik Perbandingan *Makespan* Pada Dataset SDSC

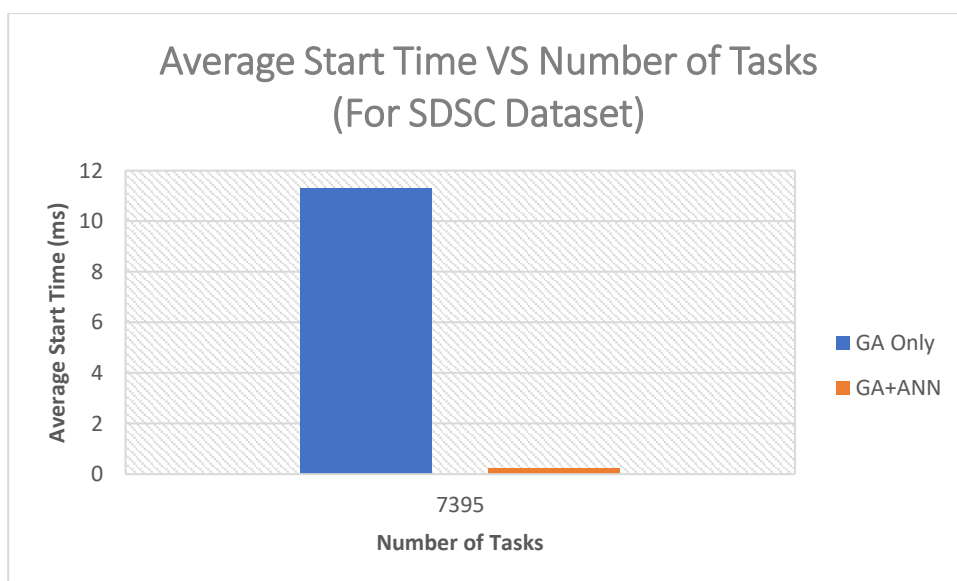
Pada dataset SDSC di Gambar 5.2, hasil yang sama bisa dilihat dimana implementasi *Genetic Algorithm* bersamaan dengan *Artificial Neural Network* menghasilkan nilai *Makespan* yang lebih rendah dibandingkan dengan implementasi *Genetic Algorithm* saja. Disini bisa disimpulkan bahwa implementasi *Genetic Algorithm* bersamaan dengan *Artificial Neural Network* memang bisa menyelesaikan *Task* terakhir lebih cepat.

5.2 Average Start Time



Gambar 5.3 Grafik Perbandingan *Average Start Time* Pada Dataset Acak

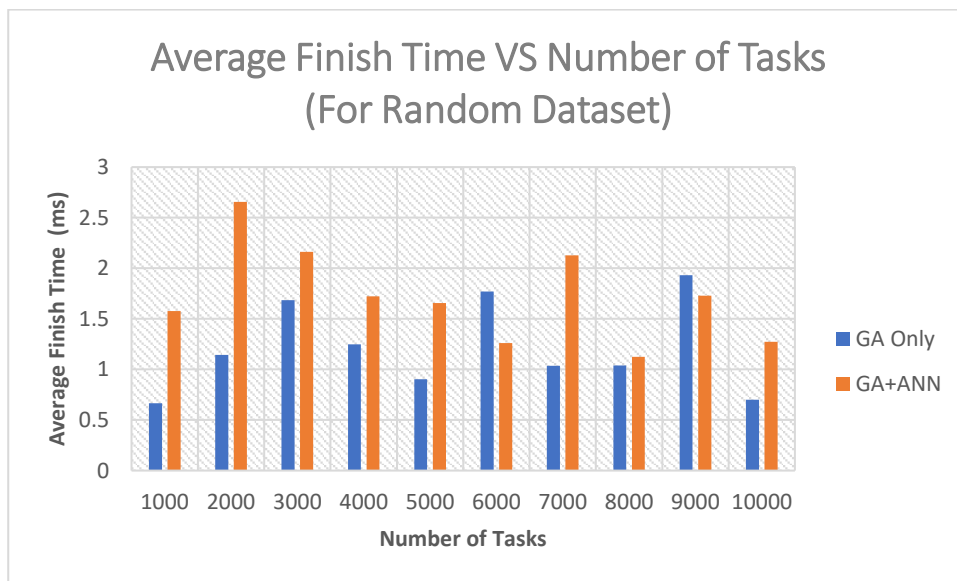
Average Start Time menggambarkan berapa lama waktu yang diperlukan untuk *Task* mulai dieksekusi di dalam prosesor dan semakin rendah nilainya maka semakin baik sebuah algoritma. Pada Gambar 5.3, bisa dilihat bahwa implementasi skenario pertama, yaitu *Genetic Algorithm* saja, bisa menghasilkan nilai yang secara konsisten lebih rendah daripada implementasi *Genetic Algorithm* bersamaan dengan *Artificial Neural Network* pada dataset acak.



Gambar 5.4 Grafik Perbandingan *Average Start Time* Pada Dataset SDSC

Namun ada hal yang menarik ketika melihat Gambar 5.4. Disini terlihat bahwa *Average Start Time* implementasi skenario kedua, yaitu *Genetic Algorithm* bersamaan dengan *Artificial Neural Network* menghasilkan nilai yang lebih rendah. Disini berarti bahwa implementasi skenario kedua bisa memulai pemrosesan *Task* lebih cepat daripada implementasi skenario pertama. Hal ini diakibatkan lebih besar dan beragamnya rentang data dari dataset SDSC dibandingkan dengan dataset acak. Dari sini bisa disimpulkan bahwa *Genetic Algorithm* lebih baik saat menghadapi dataset yang rentang dan ragam datanya tidak terlalu besar dan *Genetic Algorithm* bersamaan dengan *Artificial Neural Network* bisa digunakan untuk menghadapi dataset yang rentang dan ragam datanya besar.

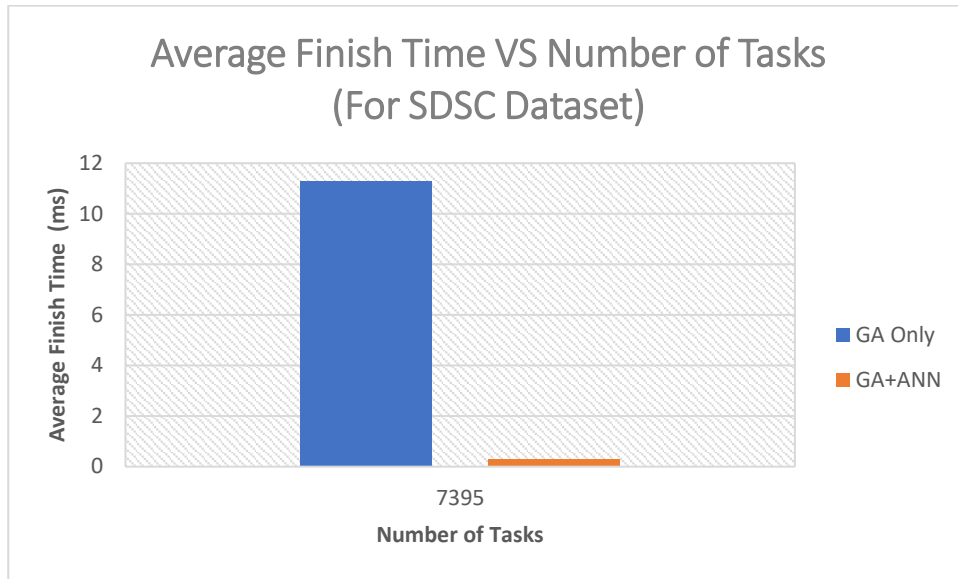
5.3 Average Finish Time



Gambar 5.5 Grafik Perbandingan *Average Finish Time* Pada Dataset Acak

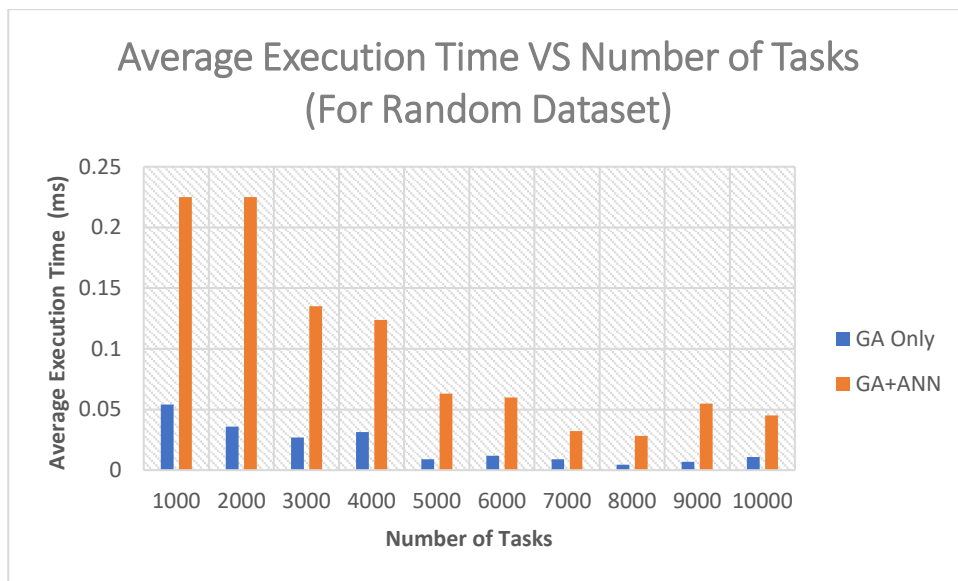
Average Finish Time berarti rata-rata waktu *Task* terselesaikan dan semakin rendah nilainya maka semakin baik sebuah algoritma. Pada Gambar 5.5, bisa dilihat bahwa implementasi skenario pertama, yaitu *Genetic Algorithm* saja, bisa menghasilkan nilai yang secara konsisten lebih rendah daripada implementasi *Genetic Algorithm* bersamaan dengan *Artificial Neural Network* pada dataset acak. Hal ini berarti implementasi skenario pertama bisa menyelesaikan pemrosesan *Task* lebih cepat daripada implementasi skenario kedua.

Namun ada hal yang menarik ketika melihat Gambar 5.6. Disini terlihat bahwa *Average Finish Time* implementasi skenario kedua, yaitu *Genetic Algorithm* bersamaan dengan *Artificial Neural Network* menghasilkan nilai yang lebih rendah. Disini berarti bahwa implementasi skenario kedua bisa menyelesaikan pemrosesan *Task* lebih cepat daripada implementasi skenario pertama. Hal ini diakibatkan lebih besar dan beragamnya rentang data dari dataset SDSC dibandingkan dengan dataset acak. Dari sini bisa disimpulkan bahwa *Genetic Algorithm* lebih baik saat menghadapi dataset yang rentang dan ragam datanya tidak terlalu besar dan *Genetic Algorithm* bersamaan dengan *Artificial Neural Network* bisa digunakan untuk menghadapi dataset yang rentang dan ragam datanya besar.



Gambar 5.6 Grafik Perbandingan *Average Finish Time* Pada Dataset SDSC

5.4 Average Execution Time

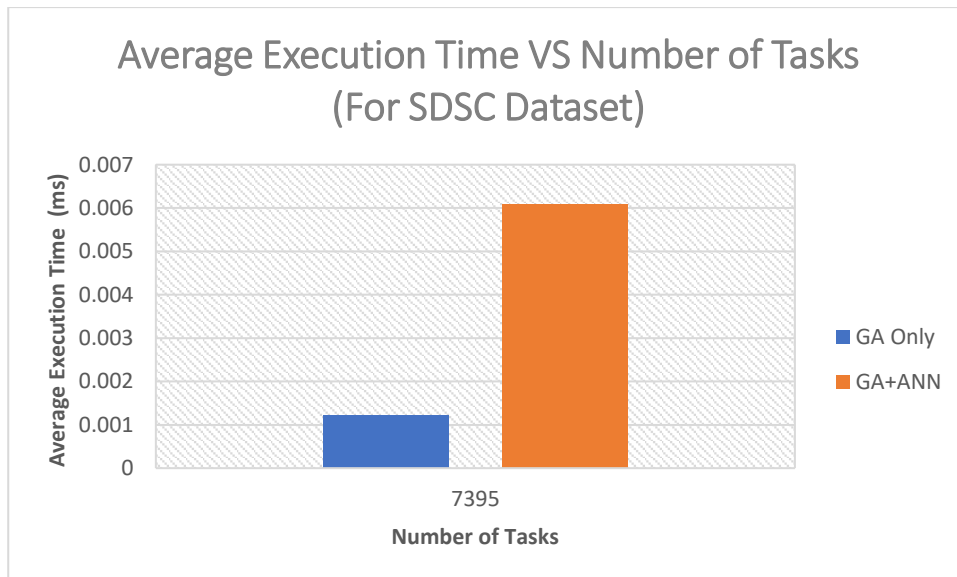


Gambar 5.7 Grafik Perbandingan *Average Execution Time* Pada Dataset Acak

Average Execution Time berarti rata-rata waktu yang *Task* habiskan di dalam pemrosesan dan semakin rendah nilainya maka semakin baik sebuah algoritma. Pada Gambar 5.7, bisa dilihat bahwa implementasi skenario pertama, yaitu *Genetic Algorithm* saja, bisa menghasilkan nilai yang secara konsisten lebih rendah daripada implementasi *Genetic Algorithm* bersamaan dengan *Artificial Neural Network* pada dataset acak. Hal ini berarti implementasi skenario pertama lebih baik dalam menghemat waktu yang *Task* habiskan saat pemrosesan daripada implementasi skenario kedua.

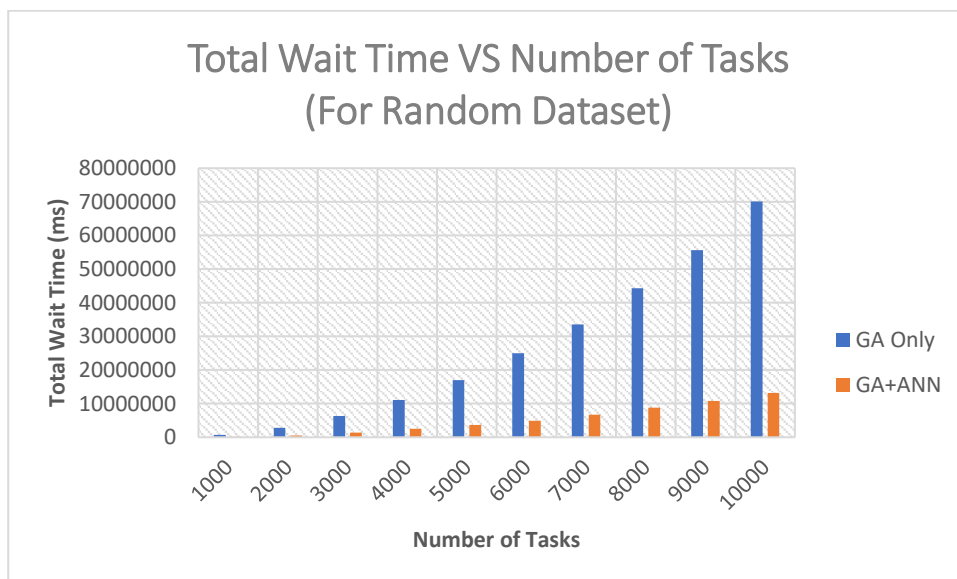
Pada dataset SDSC di Gambar 5.8, hasil yang sama bisa dilihat dimana implementasi *Genetic Algorithm* saja menghasilkan nilai *Average Finish Time* yang lebih rendah dibandingkan dengan implementasi *Genetic Algorithm* bersamaan dengan *Artificial Neural*

Network. Disini bisa disimpulkan bahwa implementasi *Genetic Algorithm* saja lebih baik dalam menghemat waktu yang *Task* habiskan saat pemrosesan daripada implementasi *Genetic Algorithm* bersamaan dengan *Artificial Neural Network*.



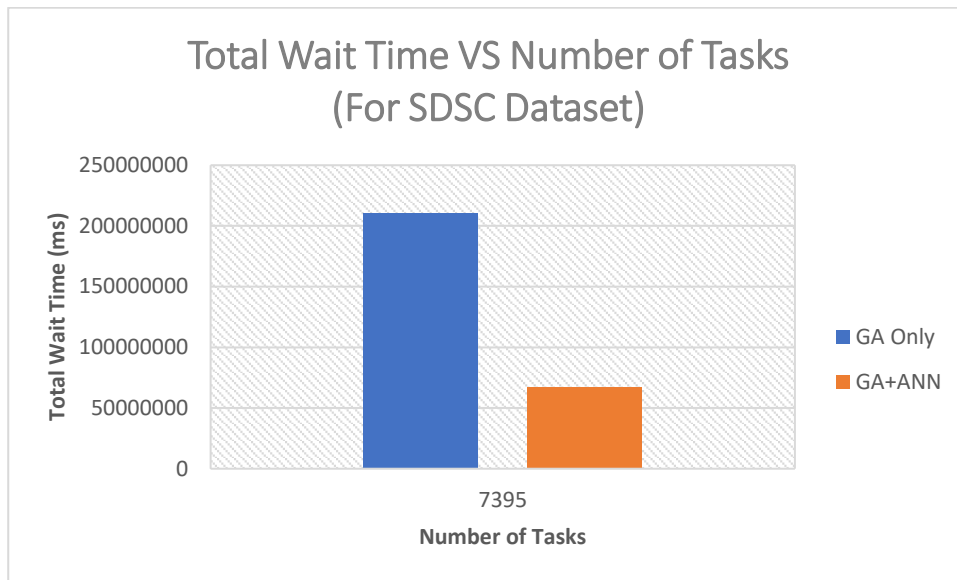
Gambar 5.8 Grafik Perbandingan *Average Execution Time* Pada Dataset SDSC

5.5 Total Wait Time



Gambar 5.9 Grafik Perbandingan *Total Wait Time* Pada Dataset Acak

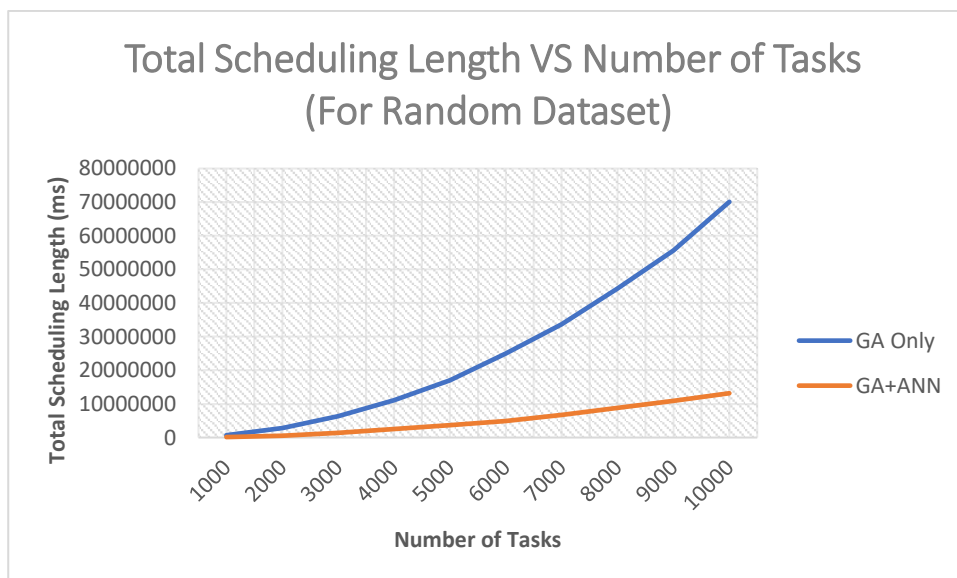
Total Wait Time berarti total waktu *Delay* yang *Task* harus terima sebelum dilakukan pemrosesan dan semakin rendah nilainya maka semakin baik sebuah algoritma. Pada Gambar 5.9, bisa dilihat bahwa implementasi skenario kedua, yaitu *Genetic Algorithm* bersamaan dengan *Artificial Neural Network*, bisa menghasilkan nilai yang secara konsisten lebih rendah daripada implementasi *Genetic Algorithm* saja pada dataset acak. Hal ini berarti implementasi skenario kedua lebih baik dalam mengurangi waktu *Delay* yang *Task* harus terima sebelum dilakukan pemrosesan daripada implementasi skenario kedua.



Gambar 5.10 Grafik Perbandingan *Total Wait Time* Pada Dataset SDSC

Pada dataset SDSC di Gambar 5.10, hasil yang sama bisa dilihat dimana implementasi *Genetic Algorithm* bersamaan dengan *Artificial Neural Network* menghasilkan nilai *Total Wait Time* yang lebih rendah dibandingkan dengan implementasi *Genetic Algorithm* saja. Disini bisa disimpulkan bahwa implementasi *Genetic Algorithm* bersamaan dengan *Artificial Neural Network* memang lebih baik dalam mengurangi waktu *Delay* yang *Task* harus terima sebelum dilakukan pemrosesan daripada implementasi *Genetic Algorithm* saja.

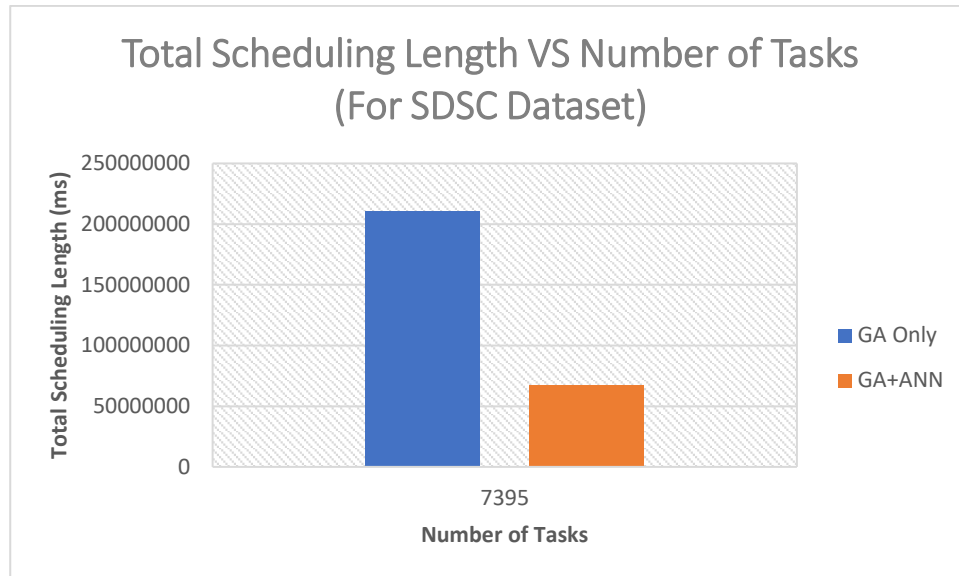
5.6 Scheduling Length



Gambar 5.11 Grafik Perbandingan *Scheduling Length* Pada Dataset Acak

Scheduling Length berarti total waktu yang diperlukan untuk bisa menyelesaikan simulasi *Cloud* dan semakin rendah nilainya maka semakin baik sebuah algoritma. Pada Gambar 5.11, bisa dilihat bahwa implementasi skenario kedua, yaitu *Genetic Algorithm* bersamaan dengan

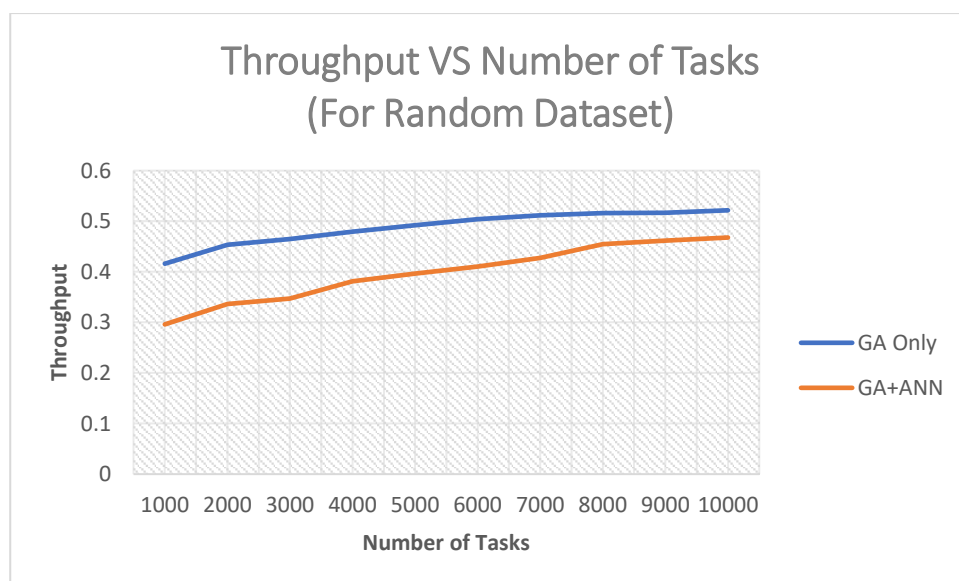
Artificial Neural Network, bisa menghasilkan nilai yang secara konsisten lebih rendah daripada implementasi *Genetic Algorithm* saja pada dataset acak. Hal ini berarti implementasi skenario kedua lebih baik dalam menghemat waktu simulasi *Cloud* daripada implementasi skenario kedua.



Gambar 5.12 Grafik Perbandingan *Scheduling Length* Pada Dataset SDSC

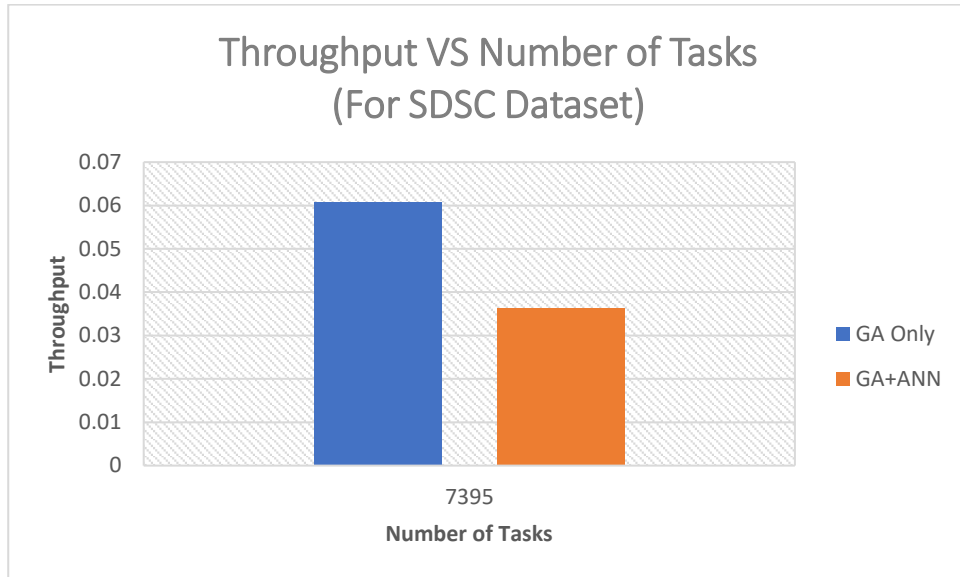
Pada dataset SDSC di Gambar 5.12, hasil yang sama bisa dilihat dimana implementasi *Genetic Algorithm* bersamaan dengan *Artificial Neural Network* menghasilkan nilai *Scheduling Length* yang lebih rendah dibandingkan dengan implementasi *Genetic Algorithm* saja. Disini bisa disimpulkan bahwa implementasi *Genetic Algorithm* bersamaan dengan *Artificial Neural Network* memang lebih baik dalam menghemat waktu simulasi *Cloud* daripada implementasi *Genetic Algorithm* saja.

5.7 Throughput



Gambar 5.13 Grafik Perbandingan *Throughput* Pada Dataset Acak

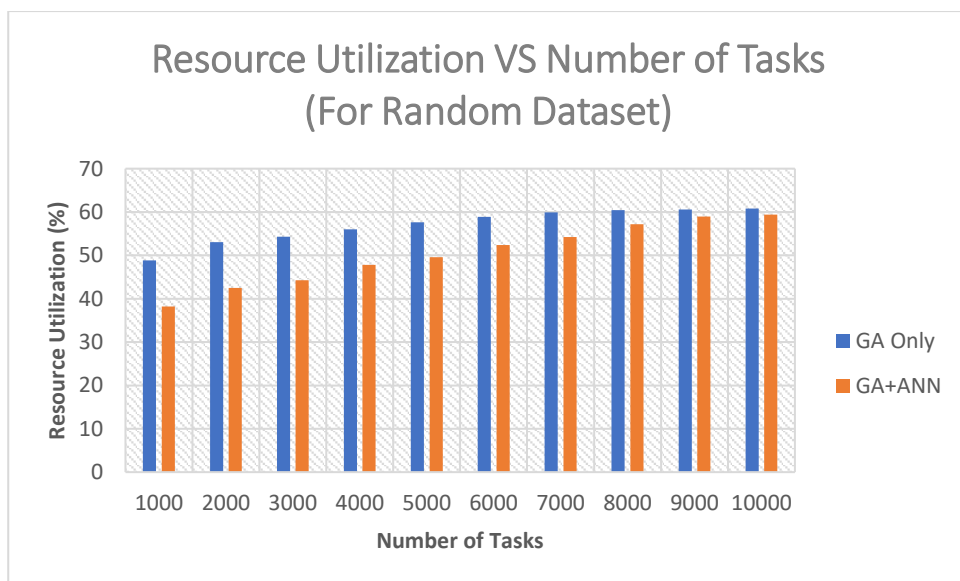
Throughput berarti banyaknya *Task* yang diselesaikan per satuan waktu dan semakin tinggi nilainya maka semakin baik sebuah algoritma. Pada Gambar 5.13, bisa dilihat bahwa implementasi skenario pertama, yaitu *Genetic Algorithm* saja, bisa menghasilkan nilai yang secara konsisten lebih tinggi daripada implementasi *Genetic Algorithm* bersamaan dengan *Artificial Neural Network* pada dataset acak.



Gambar 5.14 Grafik Perbandingan *Throughput* Pada Dataset SDSC

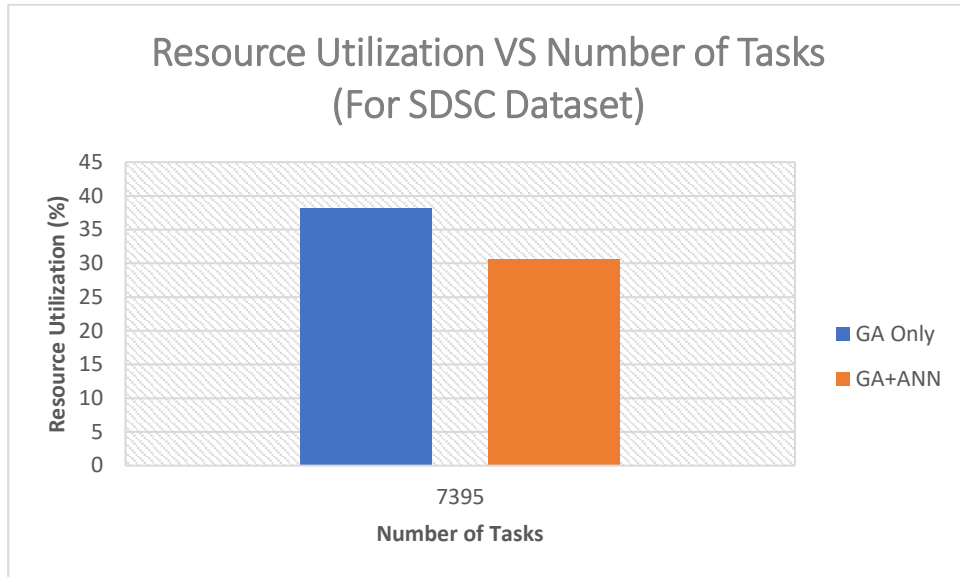
Pada dataset SDSC di Gambar 5.14, hasil yang sama bisa dilihat dimana implementasi *Genetic Algorithm* saja menghasilkan nilai *Throughput* yang lebih tinggi dibandingkan dengan implementasi *Genetic Algorithm* bersamaan dengan *Artificial Neural Network*. Disini bisa disimpulkan bahwa implementasi *Genetic Algorithm* saja memang lebih banyak menyelesaikan *Task* per satuan waktu.

5.8 Resource Utilization



Gambar 5.15 Grafik Perbandingan *Resource Utilization* Pada Dataset Acak

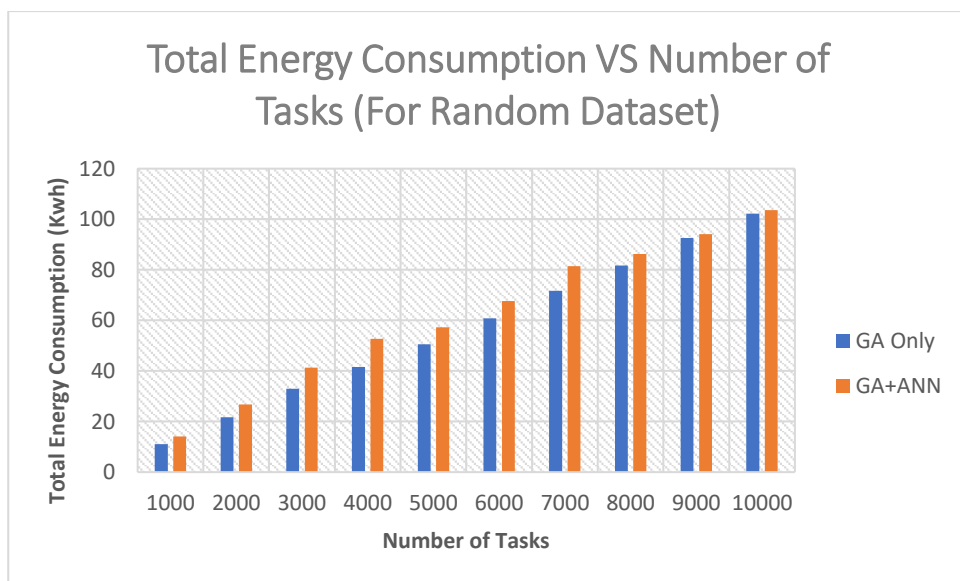
Resource Utilization berarti persentase pemanfaatan sumber daya *Cloud* dan semakin tinggi nilainya maka semakin baik sebuah algoritma. Pada Gambar 5.15, bisa dilihat bahwa implementasi skenario pertama, yaitu *Genetic Algorithm* saja, bisa menghasilkan nilai yang secara konsisten lebih tinggi daripada implementasi *Genetic Algorithm* bersamaan dengan *Artificial Neural Network* pada dataset acak.



Gambar 5.16 Grafik Perbandingan *Resource Utilization* Pada Dataset SDSC

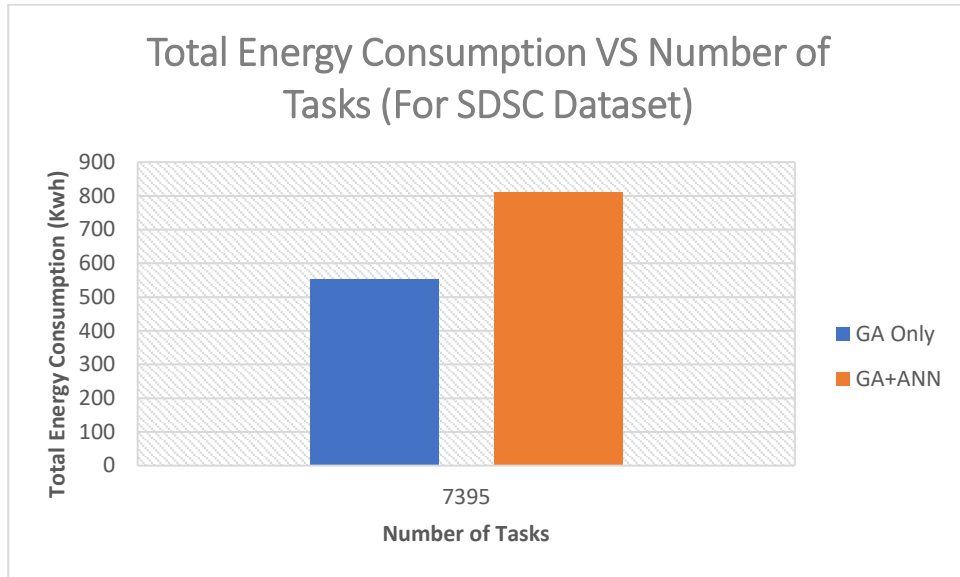
Pada dataset SDSC di Gambar 5.16, hasil yang sama bisa dilihat dimana implementasi *Genetic Algorithm* saja menghasilkan nilai *Resource Utilization* yang lebih tinggi dibandingkan dengan implementasi *Genetic Algorithm* bersamaan dengan *Artificial Neural Network*. Disini bisa disimpulkan bahwa implementasi *Genetic Algorithm* saja memang lebih baik dalam dalam persentase pemanfaatan sumber daya *Cloud*.

5.9 Total Energy Consumption



Gambar 5.17 Grafik Perbandingan *Total Energy Consumption* Pada Dataset Acak

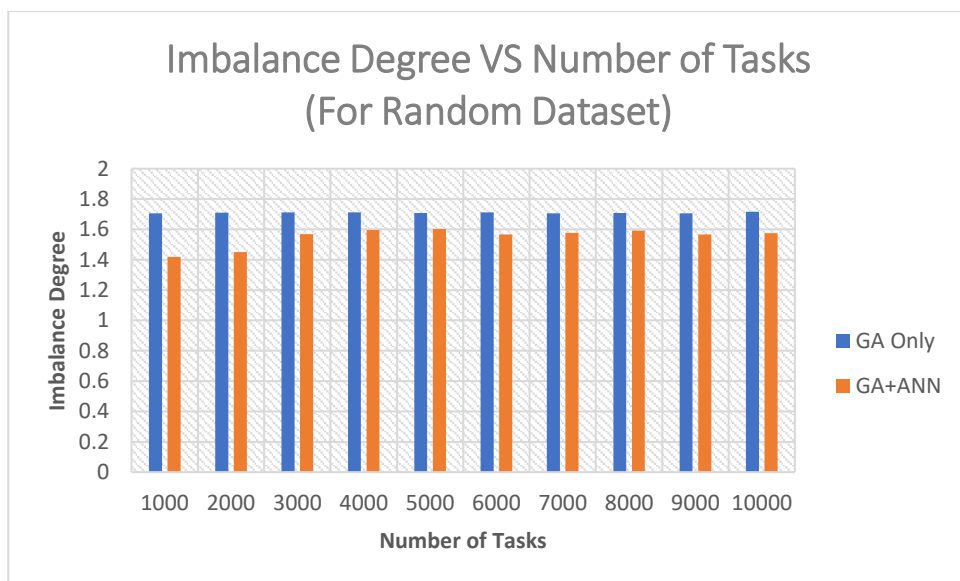
Total Energy Consumption berarti persentase besar energi yang dibutuhkan oleh sumber daya *Cloud* untuk bisa menjalankan simulasi dan semakin rendah nilainya maka semakin baik sebuah algoritma. Pada Gambar 5.17, bisa dilihat bahwa implementasi skenario pertama, yaitu *Genetic Algorithm* saja, bisa menghasilkan nilai yang secara konsisten lebih rendah daripada implementasi skenario kedua.



Gambar 5.18 Grafik Perbandingan *Total Energy Consumption* Pada Dataset SDSC

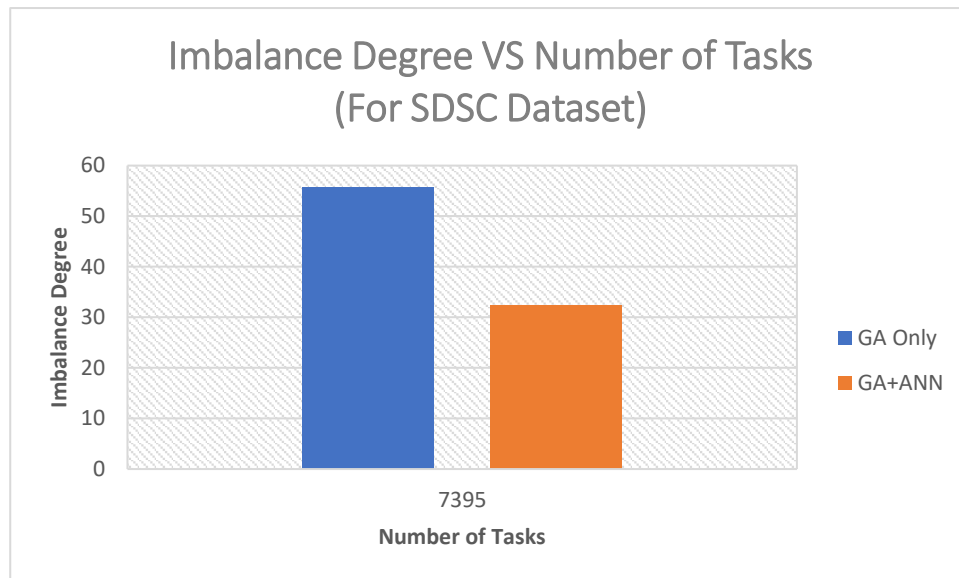
Pada dataset SDSC di Gambar 5.18, hasil yang sama bisa dilihat dimana implementasi *Genetic Algorithm* saja menghasilkan nilai *Total Energy Consumption* yang lebih rendah dibandingkan dengan implementasi *Genetic Algorithm* bersamaan dengan *Artificial Neural Network*. Disini bisa disimpulkan bahwa implementasi *Genetic Algorithm* saja memang lebih baik dalam menghemat penggunaan energi.

5.10 Imbalance Degree



Gambar 5.19 Grafik Perbandingan *Imbalance Degree* Pada Dataset Acak

Imbalance Degree berarti pengukuran ketidakseimbangan pemrosesan panjang *Task* dan semakin rendah nilainya maka semakin baik sebuah algoritma. Pada Gambar 5.19, bisa dilihat bahwa implementasi skenario kedua, yaitu *Genetic Algorithm* bersamaan dengan *Artificial Neural Network*, bisa menghasilkan nilai yang secara konsisten lebih rendah daripada implementasi *Genetic Algorithm* saja pada dataset acak. Hal ini berarti implementasi skenario kedua lebih baik dalam mengatasi ketidakseimbangan data daripada implementasi skenario pertama.



Gambar 5.20 Grafik Perbandingan *Imbalance Degree* Pada Dataset SDSC

Pada dataset SDSC di Gambar 5.20, hasil yang sama bisa dilihat dimana implementasi *Genetic Algorithm* bersamaan dengan *Artificial Neural Network* menghasilkan nilai *Imbalance Degree* yang lebih rendah dibandingkan dengan implementasi *Genetic Algorithm* saja. Disini bisa disimpulkan bahwa implementasi *Genetic Algorithm* bersamaan dengan *Artificial Neural Network* memang lebih baik dalam mengatasi ketidakseimbangan data daripada implementasi *Genetic Algorithm* saja.

BAB VI

KESIMPULAN DAN SARAN

Pada bab ini akan dijelaskan mengenai kesimpulan dari hasil uji coba yang telah dilakukan serta saran-saran tentang pengembangan yang dapat dilakukan terhadap penelitian ini di masa yang akan datang.

6.1 Kesimpulan

Berdasarkan hasil uji coba telah diberikan dan dijabarkan pada bagian-bagian sebelumnya, dapat ditarik kesimpulan bahwa:

1. Implementasi *Genetic Algorithm* bisa menghasilkan tingkat *Resource Utilization* sekitar 48% hingga 60% yang lebih tinggi daripada sistem penjadwalan tanpa menggunakan *Genetic Algorithm* yang memiliki tingkat *Resource Utilization* sekitar 30% hingga 42%
2. Implementasi *Genetic Algorithm* bersamaan dengan *Artificial Neural Network* bisa menghasilkan tingkat *Resource Utilization* sekitar 38% hingga 59% yang lebih tinggi daripada sistem penjadwalan tanpa menggunakan *Genetic Algorithm* yang memiliki tingkat *Resource Utilization* sekitar 30% hingga 42%
3. Implementasi *Genetic Algorithm* bisa menghasilkan tingkat *Resource Utilization* yang lebih tinggi daripada implementasi *Genetic Algorithm* bersamaan dengan *Artificial Neural Network*.
4. Implementasi *Genetic Algorithm* bisa melakukan tingkat penggunaan energi yang lebih rendah daripada implementasi *Genetic Algorithm* bersamaan dengan *Artificial Neural Network* dengan perbedaan sebesar 9.3%.
5. Implementasi *Genetic Algorithm* saja bisa menyelesaikan lebih banyak *Task* per satuan waktu dengan perbedaan sebesar 18.4% dan lebih baik dalam menghemat waktu yang *Task* habiskan di dalam prosesor dengan perbedaan sebesar 79.8% daripada implementasi *Genetic Algorithm* bersamaan dengan *Artificial Neural Network*
6. Implementasi *Genetic Algorithm* bersamaan dengan *Artificial Neural Network* bisa memulai pemrosesan data dengan *Delay* yang lebih sedikit dengan perbedaan sebesar 80.2% dan menyelesaikan simulasi *Cloud* dengan lebih cepat dengan perbedaan sebesar 80.2% daripada implementasi *Genetic Algorithm* saja
7. Implementasi *Genetic Algorithm* bersamaan dengan *Artificial Neural Network* lebih baik dalam menangani ketidakseimbangan data daripada implementasi *Genetic Algorithm* saja dengan perbedaan sebesar 9.3%.
8. Jika anda ingin menghemat penggunaan energi, memaksimalkan *Resource Utilization*, dan mengurangi *Execution Time* maka gunakan *Genetic Algorithm* saja.
9. Jika anda memiliki banyak *Task* dengan ketidakseimbangan data yang besar dan ingin cepat melakukan *Cloud Provisioning* maka gunakan *Genetic Algorithm* bersamaan dengan *Artificial Neural Network*.

6.2 Saran

Berikut merupakan beberapa saran untuk pengembangan dan penelitian lebih lanjut, terutama pada sistem *Cloud Provisioning*:

1. Bisa digunakan dataset lain selain milik *The San Diego Supercomputer Center (SDSC) Blue Horizon logs* karena dataset ini memiliki banyak *record data* yang hilang atau tidak tercatat sehingga susah untuk diimplementasikan.
2. Bisa digunakan sistem pembelajaran data lain selain *Artificial Neural Network* yang bisa menerima masukan berupa *array-multi-dimensi*.
3. Bisa digunakan Bahasa pemrograman lain selain *Java* yang mendukung implementasi *Random State* untuk membantu mengurangi keacakan dalam penggunaan algoritma *Genetic Algorithm* maupun penggunaan *Artificial Neural Network*.

DAFTAR PUSTAKA

- Ahmadreza Montazerolghaem, M. H.-G. (2020). Green Cloud Multimedia Networking: NFV/SDN Based Energy-Efficient Resource Allocation. *IEEE*, 4(3), 873 - 889.
- AWS. (2013, March 19). *What is cloud computing?* (AWS) Retrieved July 7, 2022, from <https://aws.amazon.com/what-is-cloud-computing/>
- Bansal, S. (2017, March 6). *How to Generate Random Numbers in Excel*. Retrieved from TrumpEXCEL: <https://trumpexcel.com/generate-random-numbers-excel/>
- Dantzig, T. (2007). *Number : the language of science (The Masterpiece Science ed.)*. New York: Plume Book.
- Education, I. C. (2021, September 3). *What Is Resource Utilization?* Retrieved from IBM: <https://www.ibm.com/cloud/blog/what-is-resource-utilization>
- Farouk A. Emara, A. A.-E. (2021). Genetic-Based Multi-objective Task Scheduling Algorithm in Cloud Computing. *International Journal of Intelligent Engineering & Systems*, 1-12.
- Foundation, T. E. (2001, November 29). *Eclipse Desktops & Web IDEs*. (The Eclipse Foundation) Retrieved July 26, 2022, from Eclipse Desktops & Web IDEs: <https://www.eclipse.org/ide/>
- G. B. Mathews, M. (1896). On the Partition of Numbers. *Proceedings of the London Mathematical Society*, s1-28(1), 486–490.
- GeeksForGeeks. (2022, June 28). *Virtualization In Cloud Computing and Types*. Retrieved from GeeksForGeeks: <https://www.geeksforgeeks.org/virtualization-cloud-computing-types/>
- Hardesty, L. (2017, April 4). *Explained: Neural networks*. (MIT News Office) Retrieved July 25, 2022, from <https://news.mit.edu/2017/explained-neural-networks-deep-learning-0414>
- Heaton, J. (2011). *Programming Neural Networks with Encog3 in Java*. Saint Louis: Heaton Research, Inc.
- Heaton, J. (2015). Encog: Library of Interchangeable Machine Learning Models. *Journal of Machine Learning Research*, 1-5.
- Heaton, J. (2022, 12 25). *Encog Machine Learning Framework*. Retrieved from Heaton Research: <https://www.heatonresearch.com/encog/>
- Henning Titi Ciptaningtyas, A. M. (2022). Survey on Task Scheduling Methods. 2022 *International Seminar on Intelligent Technology and Its Applications (ISITIA)*. Surabaya: IEEE.

- Kalita, D. (2022, April 6). *An Overview and Applications of Artificial Neural Networks*. (Analytics Vidhya) Retrieved July 22, 2022, from <https://www.analyticsvidhya.com/blog/2022/03/an-overview-and-applications-of-artificial-neural-networks-ann>
- Knuth, D. (1968). *The Art of Computer Programming*. United States: Stanford CS.
- Kwangwoog Jung, Y.-K. C.-J. (2017). Performance evaluation of ROMS v3.6 on a commercial cloud system. *ResearchGate*.
- Laboratory, C. C. (2018). *CloudSim: A Framework For Modeling And Simulation Of Cloud Computing Infrastructures And Services*. (School of Computing and Information Systems) Retrieved July 26, 2022, from Laboratory, Cloud Computing and Distributed Systems (CLOUDS): <http://www.cloudbus.org/cloudsim/>
- Larasati, K. D. (2019, July 9). *Artificial Neural Network*. (Medium) Retrieved July 25, 2022, from <https://medium.com/@dhea.larasati326/artificial-neural-network-55797915f14a>
- log, T. S. (2003, January). *The San Diego Supercomputer Center (SDSC) Blue Horizon log*. (The San Diego Supercomputer Center (SDSC)) Retrieved July 22, 2022, from https://www.cs.huji.ac.il/labs/parallel/workload/l_sdsc_blue/index.html
- Martin Feuerman, H. W. (1973). A Mathematical Programming Model for Test Construction and Scoring. *Inform*, 19(8), 841-971.
- Meier, K. (2020, June 1). *What Resource Utilization Is and How To Calculate It*. Retrieved from Float Resources: <https://www.float.com/resources/guide-to-resource-utilization/>
- Michael Pawlish, A. S. (2012, June). Analyzing Utilization Rates in Data Centers for Optimizing Energy. *2012 International Green Computing Conference (IGCC)* (pp. 1-6). San Jose: IEEE.
- Michael Pawlish, A. S. (2014). A Call for Energy Efficiency in Data Centers. *ACM SIGMOD Record*, 45-51.
- Mitchell, M. (1996). *An Introduction to Genetic Algorithms*. Cambridge, MA: MIT Press.
- Mohammad Hamdaqa, L. T. (2012). Cloud Computing Uncovered: A Research Landscape. *Elsevier*, 86, 41-86.
- Montgomery, J. (2020, October). *Cloud Provisioning*. (Tech Target) Retrieved July 22, 2021, from <https://www.techtarget.com/searchitchannel/definition/cloud-provisioning>
- Moore, S. (2019, July 5). *How to Avoid Overloading Your IT Project Team*. Retrieved from Gartner: <https://www.gartner.com/smarterwithgartner/avoid-overloading-project-team>
- Pradeep Singh Rawat, P. D. (2020). Resource provisioning in scalable cloud using bio-inspired artificial neural network model. *Elsevier*, 1-16.
- Ray, P. P. (2017). An Introduction to Dew Computing: Definition, Concept and Implications. *IEEE*, 6, 723-737.

- Sharma, S. (2017, September 6). *Activation Functions in Neural Networks*. Retrieved from Medium: <https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6>
- Singh, A. (2019, July 10). *Beginners Guide to Cloudsim Project Structure*. Retrieved from Cloudsim Tutorials: <https://www.cloudsimtutorials.online/beginners-guide-to-cloudsim-project-structure/>
- Steve Chapin, J. P. (2000). Benchmarks and Standards for the Evaluation of Parallel Job Schedulers. *ResearchGate*, July.
- Suryansh. (2018, March 26). *Genetic Algorithms + Neural Networks = Best of Both Worlds*. (Towards Data Science) Retrieved July 25, 2022, from <https://towardsdatascience.com/gas-and-nns-6a41f1e8146d>
- Technology, N. I. (2011). The NIST Definition of Cloud Computing. *National Institute of Standards and Technology*, 1-7.
- Terh, F. (2019, March 28). *How to solve the Knapsack Problem with dynamic programming*. (Medium) Retrieved July 7, 2022, from <https://medium.com/@fabianterh/how-to-solve-the-knapsack-problem-with-dynamic-programming-eb88c706d3cf>
- Whitley, D. (1994). A Genetic Algorithm Tutorial. *Statistics and Computing*, 1-37.
- Wray, J. (2014, February 27). *Where's The Rub: Cloud Computing's Hidden Costs*. (Forbes) Retrieved July 21, 2022, from <https://www.forbes.com/sites/centurylink/2014/02/27/wheres-the-rub-cloud-computings-hidden-costs/>

LAMPIRAN

A. Kode Sumber *CloudSimulationGA.java*

```
package org.cloudbus.cloudsim.examples;

import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.FileWriter;
import java.text.DecimalFormat;
import java.util.ArrayList;
import java.util.Calendar;
import java.util.DoubleSummaryStatistics;
import java.util.LinkedList;
import java.util.List;
import java.util.Scanner;
import java.util.stream.DoubleStream;

import org.cloudbus.cloudsim.Cloudlet;
import org.cloudbus.cloudsim.CloudletSchedulerSpaceShared;
import org.cloudbus.cloudsim.DatacenterBroker;
import org.cloudbus.cloudsim.DatacenterCharacteristics;
import org.cloudbus.cloudsim.Log;
import org.cloudbus.cloudsim.Pe;
import org.cloudbus.cloudsim.Storage;
import org.cloudbus.cloudsim.UtilizationModel;
import org.cloudbus.cloudsim.UtilizationModelFull;
import org.cloudbus.cloudsim.Vm;
import org.cloudbus.cloudsim.VmSchedulerTimeShared;
import org.cloudbus.cloudsim.core.CloudSim;
import org.cloudbus.cloudsim.power.PowerHost;
import org.cloudbus.cloudsim.power.PowerDatacenter;
import org.cloudbus.cloudsim.power.PowerHostUtilizationHistory;
import org.cloudbus.cloudsim.power.PowerVmAllocationPolicySimple;
import org.cloudbus.cloudsim.power.models.PowerModelLinear;
import org.cloudbus.cloudsim.provisioners.BwProvisionerSimple;
import org.cloudbus.cloudsim.provisioners.PeProvisionerSimple;
import org.cloudbus.cloudsim.provisioners.RamProvisionerSimple;

public class CloudSimulationGA {

    private static PowerDatacenter datacenter1, datacenter2, datacenter3, datacenter4, datacenter5, datacenter6;
    /** The cloudlet list. */
    private static List<Cloudlet> cloudletList;

    /** The vmList. */
    private static List<Vm> vmList;

    private static List<Vm> createVM(int userId, int vms) {
```

```

//Creates a container to store VMs.
//This list is passed to the broker later
LinkedList<Vm> list = new LinkedList<Vm>();

//VM Parameters
long size = 10000; //Image size (MB)
int[] ram = {512,1024,2048}; //VM memory (MB)
int[] mips = {400,500,600}; //VM processing power (MIPS)
long bw = 1000; //VM bandwidth
int pesNumber = 1; //Number of cpus
String vmm = "Xen"; //VMM name

//create VMs
Vm[] vm = new Vm[vms];

for(int i=0;i<vms;i++){
    //For loop to create a VM with a time shared scheduling policy for
cloudlets:
    vm[i] = new Vm(i, userId, mips[i%3], pesNumber, ram[i%3], bw, size,
vmm, new CloudletSchedulerSpaceShared());
    list.add(vm[i]);
}

return list;
}

private static ArrayList<Double> getSeedValue(int cloudletcount){

    // Creating an arraylist to store Cloudlet Datasets
    ArrayList<Double> seed = new ArrayList<Double>();
    Log.println(System.getProperty("user.dir")+ "/SDSCDataset.txt");

    try{
        // Opening and scanning the file
        File fobj = new File(System.getProperty("user.dir")+
"/SDSCDataset.txt");

        java.util.Scanner readFile = new java.util.Scanner(fobj);

        while(readFile.hasNextLine() && cloudletcount>0)
        {
            // Adding the file to the arraylist
            seed.add(readFile.nextDouble());
            cloudletcount--;
        }
        readFile.close();

    } catch (FileNotFoundException e) {
        e.printStackTrace();
    }

    return seed;
}

private static List<Cloudlet> createCloudlet(int userId, int cloudlets){

```

```

        ArrayList<Double> randomSeed = getSeedValue(cloudlets);

        // Creates a container to store Cloudlets
        LinkedList<Cloudlet> list = new LinkedList<Cloudlet>();

        //Cloudlet parameters
        long length = 0; // Cloudlet length (MI) - 0 for SDSC
        //long length = 1000; // Cloudlet length (MI) - 1000 for Random Dataset
        long fileSize = 300; // Cloudlet file size (MB)
        long outputSize = 300; // Cloudlet file size (MB)
        int pesNumber = 1; // Cloudlet CPU needed to process
        UtilizationModel utilizationModel = new UtilizationModelFull();

        Cloudlet[] cloudlet = new Cloudlet[cloudlets];

        for(int i=0;i<cloudlets;i++){
            long finalLen = length +
Double.valueOf(randomSeed.get(i)).longValue();
            // Creating the cloudlet with all the parameter listed
            cloudlet[i] = new Cloudlet(i, finalLen, pesNumber, fileSize, outputSize,
utilizationModel, utilizationModel, utilizationModel);

            // setting the owner of these Cloudlets
            cloudlet[i].setUserId(userId);
            list.add(cloudlet[i]);
        }

        return list;
    }

    /**
    * Creates main() to run this example
    */
    public static void main(String[] args) {
        Log.println("Starting Cloud Simulation Example...");

        try {
            // First step: Initialize the CloudSim package. It should be called
            // before creating any entities.
            int num_user = 1; // Number of grid users
            Calendar calendar = Calendar.getInstance();
            boolean trace_flag = false; // Mean trace events
            int hostId=0; // Starting host ID
            BufferedWriter outputWriter = null;
            outputWriter = new BufferedWriter(new FileWriter("filename.txt"));

//Save output to text file

            int vmNumber = 54; // The number of VMs created
            int cloudletNumber = 7395; // The number of Tasks created

            // Initialize the CloudSim library
            CloudSim.init(num_user, calendar, trace_flag);

            //Second step: Create Data Centers

```


one of them to run a CloudSim simulation

```
//Datacenters are the resource providers in CloudSim. We need at least
```

```
datacenter1 = createDatacenter("DataCenter_1", hostId);  
hostId = 3;  
datacenter2 = createDatacenter("DataCenter_2", hostId);  
hostId = 6;  
datacenter3 = createDatacenter("DataCenter_3", hostId);  
hostId = 9;  
datacenter4 = createDatacenter("DataCenter_4", hostId);  
hostId = 12;  
datacenter5 = createDatacenter("DataCenter_5", hostId);  
hostId = 15;  
datacenter6 = createDatacenter("DataCenter_6", hostId);
```

```
//Third step: Create Broker
```

```
DatacenterBroker broker = createBroker();  
int brokerId = broker.getId();
```

```
//Fourth step: Create VMs and Cloudlets
```

```
vmList = createVM(brokerId,vmNumber); //Creating vms  
cloudletList = createCloudlet(brokerId,cloudletNumber); // Creating
```

cloudlets

```
//Fifth step: Send VMs and Cloudlets to broker
```

```
broker.submitVmList(vmList);  
broker.submitCloudletList(cloudletList);
```

```
//Sixth step: Use Genetic Algorithm
```

```
int chromosomeLength = 9; //number of genes inside a chromosome  
int cloudletLoopingNumber = cloudletNumber/vmNumber - 1; //number
```

of iteration needed to process the dataset

```
for (int cloudletIterator=0; cloudletIterator<=cloudletLoopingNumber;
```

cloudletIterator++)

```
{
```

```
    System.out.println("Cloudlet Iteration Number " +
```

cloudletIterator);

```
    for (int dataCenterIterator = 1; dataCenterIterator <= 6;
```

dataCenterIterator++)

```
    {
```

```
        // Initialize Genetic Algorithm
```

```
        GeneticAlgorithm ga = new GeneticAlgorithm(20, 0.3,  
0.95, 2, cloudletList, vmList);
```

```
        // Initialize population
```

```
        System.out.println("Datacenter " + dataCenterIterator +  
" Population Initialization");
```

```
        Population population =
```

```
        ga.initPopulation(chromosomeLength, dataCenterIterator);
```

```
        // Evaluate population
```

```

cloudletIterator);

// Genetic Algorithm Iteration
int iteration = 1;
while (iteration <= 15)
{
    // get fittest individual from population in
    Individual fit = population.getFittest(0);

    System.out.print("Fittest: ");
    for(int j=0;j<9;j++) {
        System.out.print(fit.chromosome[j] +
        " ");

    }
    System.out.println(" fitness => " +
    fit.getFitness());

    ga.crossoverPopulation(population, dataCenterIterator);

    // Apply crossover
    population =

    // Apply mutation
    population = ga.mutatePopulation(population,
    dataCenterIterator);

    // Evaluate population
    ga.evalPopulation(population,
    dataCenterIterator, cloudletIterator);

    // Increment the current generation
    iteration++;
}
// Get the fittest individual from Genetic Algorithm
System.out.println("Best solution of GA: " + population.getFittest(0) + " For Datacenter-" + dataCenterIterator);
System.out.println("Highest Fitness Achieved: " +
population.getFittest(0).getFitness());

// Assign Cloudlet to their respective VMs according to
the fittest individual's chromosome
//outputWriter.write("{}");
for (int assigner=0+(dataCenterIterator-1)*9 + cloudletIterator*54; assigner<9+(dataCenterIterator-1)*9 +
cloudletIterator*54; assigner++)
{
    broker.bindCloudletToVm(assigner,
    population.getFittest(0).getGene(assigner%9));

    outputWriter.write(Long.toString(population.getFittest(0).getGene(assigner%9)%9)); // Print Assigned VM ID %
    outputWriter.write(" ");
    //if (assigner%9<8)
    //{
    //    outputWriter.write(",");
    //}
}

```

```

        }
        outputWriter.newLine();
        //outputWriter.write("{}");
    }
}

// Seventh step: Starts the simulation
CloudSim.startSimulation();

outputWriter.flush();
outputWriter.close();
// Final step: Print results when simulation is over
List<Cloudlet> newList = broker.getCloudletReceivedList();

CloudSim.stopSimulation();

printCloudletList(newList);

Log.println("Cloud Simulation Example finished!");
}
catch (Exception e)
{
    e.printStackTrace();
    Log.println("The simulation has been terminated due to an unexpected
error");
}
}

private static PowerDatacenter createDatacenter(String name, int hostId){

    // Here are the steps needed to create a PowerDatacenter:
    // 1. We need to create a list to store one or more machines
    List<PowerHost> hostList = new ArrayList<PowerHost>();

    // 2. A Machine contains one or more PEs or CPUs/Cores. Therefore, should
    // create a list to store these PEs before creating a Machine.
    List<Pe> peList1 = new ArrayList<Pe>();
    List<Pe> peList2 = new ArrayList<Pe>();
    List<Pe> peList3 = new ArrayList<Pe>();

    int mipsunused= 300; // Unused core, only 3 cores will be able to process
    Cloudlets for this simulation

    int mips1 = 400; // The MIPS Must be bigger than the VMs
    int mips2 = 500;
    int mips3 = 600;

    // 3. Create PEs and add these into the list.
    //for a quad-core machine, a list of 4 PEs is required:
    peList1.add(new Pe(0, new PeProvisionerSimple(mips1))); // need to store Pe id
    and MIPS Rating, Must be bigger than the VMs

```

```

peList1.add(new Pe(1, new PeProvisionerSimple(mips1)));
peList1.add(new Pe(2, new PeProvisionerSimple(mips1)));
peList1.add(new Pe(3, new PeProvisionerSimple(mipsunused)));
peList2.add(new Pe(4, new PeProvisionerSimple(mips2)));
peList2.add(new Pe(5, new PeProvisionerSimple(mips2)));
peList2.add(new Pe(6, new PeProvisionerSimple(mips2)));
peList2.add(new Pe(7, new PeProvisionerSimple(mipsunused)));
peList3.add(new Pe(8, new PeProvisionerSimple(mips3)));
peList3.add(new Pe(9, new PeProvisionerSimple(mips3)));
peList3.add(new Pe(10, new PeProvisionerSimple(mips3)));
peList3.add(new Pe(11, new PeProvisionerSimple(mipsunused)));

//4. Create Hosts with its id and list of PEs and add them to the list of machines
int ram = 128000 ; //Host memory (MB), Must be bigger than the VMs
long storage = 1000000; //Host storage (MB)
int bw = 10000; //Host bandwidth
int maxpower = 117; // Host Max Power
int staticPowerPercentage = 50; // Host Static Power Percentage

hostList.add(
    new PowerHostUtilizationHistory(
        hostId, new RamProvisionerSimple(ram),
        new BwProvisionerSimple(bw),
        storage,
        peList1,
        new VmSchedulerTimeShared(peList1),
        new PowerModelLinear(maxpower, staticPowerPercentage)));
hostId++;

hostList.add(
    new PowerHostUtilizationHistory(
        hostId, new RamProvisionerSimple(ram),
        new BwProvisionerSimple(bw),
        storage,
        peList2,
        new VmSchedulerTimeShared(peList2),
        new PowerModelLinear(maxpower, staticPowerPercentage)));
hostId++;

hostList.add(
    new PowerHostUtilizationHistory(
        hostId, new RamProvisionerSimple(ram),
        new BwProvisionerSimple(bw),
        storage,
        peList3,
        new VmSchedulerTimeShared(peList3),
        new PowerModelLinear(maxpower, staticPowerPercentage)));

// 5. Create a DatacenterCharacteristics object that stores the
// properties of a data center: architecture, OS, list of
// Machines, allocation policy: time- or space-shared, time zone
// and its price (G$/Pe time unit).
String arch = "x86"; // System architecture

```

```

        String os = "Linux";           // Operating system
        String vmm = "Xen";             // Name
        double time_zone = 10.0;        // Time zone this resource located
        double cost = 3.0;               // The cost of using processing in this resource
        double costPerMem = 0.05;        // The cost of using memory in this
resource
        double costPerStorage = 0.1;     // The cost of using storage in this resource
        double costPerBw = 0.1;          // The cost of using bw in this
resource

        LinkedList<Storage> storageList = new LinkedList<Storage>();

        DatacenterCharacteristics characteristics = new DatacenterCharacteristics(
arch, os, vmm, hostList, time_zone, cost, costPerMem, costPerStorage,
costPerBw);

// 6. Finally, we need to create a PowerDatacenter object.
PowerDatacenter datacenter = null;
try {
    datacenter = new PowerDatacenter(name, characteristics, new
PowerVmAllocationPolicySimple(hostList), storageList, 9);
} catch (Exception e) {
    e.printStackTrace();
}

return datacenter;
}

private static DatacenterBroker createBroker(){

    DatacenterBroker broker = null;
    try {
        broker = new DatacenterBroker("Broker");
    } catch (Exception e) {
        e.printStackTrace();
        return null;
    }
    return broker;
}

/**
 * Prints the Cloudlet objects
 * @param list list of Cloudlets
 * @throws FileNotFoundException
 */
private static void printCloudletList(List<Cloudlet> list) throws FileNotFoundException {

    // Initializing the printed output to zero
    int size = list.size();
    Cloudlet cloudlet = null;

    String indent = "  ";

```

```

        Log.println();
Log.println("===== OUTPUT =====");
Log.println("Cloudlet ID" + indent + "STATUS" + indent +
    "Data center ID" + indent + "VM ID" + indent + "Time"
        + indent + "Start Time" + indent + "Finish Time" + indent +
"Waiting Time");

    double waitTimeSum = 0.0;
    double CPUTimeSum = 0.0;
    int totalValues = 0;
    DecimalFormat dft = new DecimalFormat("###.##");

    double response_time[] = new double[size];

    // Printing all the status of the Cloudlets
    for (int i = 0; i < size; i++) {
        cloudlet = list.get(i);
        Log.print(cloudlet.getCloudletId() + indent + indent);

        if (cloudlet.getCloudletStatus() == Cloudlet.SUCCESS){
            Log.print("SUCCESS");
            CPUTimeSum = CPUTimeSum + cloudlet.getActualCPUTime();
            waitTimeSum = waitTimeSum + cloudlet.getWaitingTime();
            Log.println(indent + indent + indent + (cloudlet.getResourceId()-1) + indent + indent + indent +
cloudlet.getVmId() +
                indent + indent + dft.format(cloudlet.getActualCPUTime()) + indent + indent +
dft.format(cloudlet.getExecStartTime())+
                indent + indent + dft.format(cloudlet.getFinishTime())+ indent + indent + indent +
dft.format(cloudlet.getWaitingTime()));
            totalValues++;

            response_time[i] = cloudlet.getActualCPUTime();
        }
    }

    DoubleSummaryStatistics stats = DoubleStream.of(response_time).summaryStatistics();

    // Show the parameters and print them out
    Log.println();
    System.out.println("min = " + stats.getMin());
    System.out.println("Response_Time: " + CPUTimeSum/totalValues);

    Log.println();
    Log.println("TotalCPUTime : " + CPUTimeSum);
    Log.println("TotalWaitTime : " + waitTimeSum);
    Log.println("TotalCloudletsFinished : " + totalValues);
    Log.println();
    Log.println();

    //Average Cloudlets Finished
    Log.println("AverageCloudletsFinished : " + (CPUTimeSum/ totalValues));

    //Average Start Time
    double totalStartTime =0.0;
    for (int i = 0; i < size; i++) {

```

```

        totalStartTime = cloudletList.get(i).getExecStartTime();
    }
    double avgStartTime = totalStartTime/size;
    System.out.println("Average StartTime: " + avgStartTime );

    //Average Execution Time
    double ExecTime =0.0;
    for (int i = 0; i < size; i++) {
        ExecTime = cloudletList.get(i).getActualCPUTime();
    }
    double avgExecTime = ExecTime/size;
    System.out.println("Average Execution Time: " + avgExecTime );

    //Average Finish Time
    double totalTime =0.0;
    for (int i = 0; i < size; i++) {
        totalTime = cloudletList.get(i).getFinishTime();
    }
    double avgTAT = totalTime/size;
    System.out.println("Average FinishTime: " + avgTAT );

    //Average Waiting Time
    double avgWT = cloudlet.getWaitingTime()/size;
    System.out.println("Average Waiting time: " + avgWT);

    Log.println();
    Log.println();

    //Throughput
    double maxFT =0.0;
    for (int i = 0; i < size; i++) {
        double currentFT = cloudletList.get(i).getFinishTime();
        if (currentFT > maxFT) {
            maxFT = currentFT;
        }
    }
    double throughput = size/maxFT;
    System.out.println("Throughput: " + throughput );

    //Makespan
    double makespan =0.0;
    double makespan_total = makespan + cloudlet.getFinishTime();
    System.out.println("Makespan: " + makespan_total);

    //Imbalance Degree
    double degree_of_imbalance = (stats.getMax() - stats.getMin())/(CPUTimeSum/ totalValues);
    System.out.println("Imbalance Degree: " + degree_of_imbalance);

    //Scheduling Length
    double scheduling_length = waitTimeSum + makespan_total;
    Log.println("Total Scheduling Length: " + scheduling_length);

    //CPU Resource Utilization
    double resource_utilization = (CPUTimeSum / (makespan_total * 54)) * 100;
    Log.println("Resource Utilization: " + resource_utilization);

```

```

        //Energy Consumption
        Log.println(String.format("Total Energy Consumption: %.2f kWh",
        (datacenter1.getPower() + datacenter2.getPower()+ datacenter3.getPower()+ datacenter4.getPower()+
        datacenter5.getPower()+ datacenter6.getPower())/ (3600*1000)));
    }
}

```

B. Kode Sumber *CloudSimulationANN.Java*

```

package org.cloudbus.cloudsim.examples;

import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.PrintStream;
import java.text.DecimalFormat;
import java.util.ArrayList;
import java.util.Calendar;
import java.util.DoubleSummaryStatistics;
import java.util.LinkedList;
import java.util.List;
import java.util.Scanner;
import java.util.stream.DoubleStream;

import org.cloudbus.cloudsim.Cloudlet;
import org.cloudbus.cloudsim.CloudletSchedulerSpaceShared;
import org.cloudbus.cloudsim.DatacenterBroker;
import org.cloudbus.cloudsim.DatacenterCharacteristics;
import org.cloudbus.cloudsim.Log;
import org.cloudbus.cloudsim.Pe;
import org.cloudbus.cloudsim.Storage;
import org.cloudbus.cloudsim.UtilizationModel;
import org.cloudbus.cloudsim.UtilizationModelFull;
import org.cloudbus.cloudsim.Vm;
import org.cloudbus.cloudsim.VmSchedulerTimeShared;
import org.cloudbus.cloudsim.core.CloudSim;
import org.cloudbus.cloudsim.power.PowerHost;
import org.cloudbus.cloudsim.power.PowerDatacenter;
import org.cloudbus.cloudsim.power.PowerHostUtilizationHistory;
import org.cloudbus.cloudsim.power.PowerVmAllocationPolicySimple;
import org.cloudbus.cloudsim.power.models.PowerModelLinear;
import org.cloudbus.cloudsim.provisioners.BwProvisionerSimple;
import org.cloudbus.cloudsim.provisioners.PeProvisionerSimple;
import org.cloudbus.cloudsim.provisioners.RamProvisionerSimple;
import org.encog.ml.data.MLData;
import org.encog.ml.data.MLDataPair;

```



```

import org.encog.ml.data.MLDataSet;
import org.encog.ml.data.basic.BasicMLData;
import org.encog.ml.data.basic.BasicMLDataSet;
import org.encog.neural.networks.BasicNetwork;
import org.encog.neural.networks.training.lma.LevenbergMarquardtTraining;
import org.encog.neural.networks.training.propagation.back.Backpropagation;
import org.encog.neural.networks.training.propagation.manhattan.ManhattanPropagation;
import org.encog.neural.networks.training.propagation.quick.QuickPropagation;
import org.encog.neural.networks.training.propagation.resilient.ResilientPropagation;
import org.encog.neural.networks.training.propagation.scg.ScaledConjugateGradient;
import org.encog.persist.EncogDirectoryPersistence;
import org.encog.util.arrayutil.NormalizationAction;
import org.encog.util.arrayutil.NormalizedField;

public class CloudSimulationANN {

// Double Array to hold the raw length data
public static double LENGTH_RAW_DATA[][];

// Double Array to hold the raw target data
public static double TARGET_RAW_DATA[][];

public static double[][] Reading2DArrayFromFileLength()
{
    Scanner scannerLength;
    int rows = 163; // Number of rows to be scanned
    int columns = 9; // Number of columns to be scanned
    double [][] arrayLength = new double[rows][columns];

    try
    {
        scannerLength = new Scanner(new BufferedReader(new FileReader(System.getProperty("user.dir")+
        "/test/TestLength-SDSC.txt")));

        while(scannerLength.hasNextLine()) {
            for (int i=0; i<arrayLength.length; i++) {
                String[] line = scannerLength.nextLine().trim().split(" "); //
                for (int j=0; j<line.length; j++){
                    arrayLength[i][j] = Integer.parseInt(line[j]); // Parsing
                }
            }
        }
    } catch (FileNotFoundException e)
    {
        e.printStackTrace();
    }
    return arrayLength;
}

public static double[][] Reading2DArrayFromFileTarget()
{
    Scanner scannerTarget;
    int rows = 163; // Number of rows to be scanned
    int columns = 9; // Number of columns to be scanned

```

```

        double [][] arrayTarget = new double[rows][columns];

        try
        {
scannerTarget = new Scanner(new BufferedReader(new FileReader(System.getProperty("user.dir")+ "/test/TestTarget-
SDSC.txt")));

                while(scannerTarget.hasNextLine()) {
                    for (int i=0; i<arrayTarget.length; i++) {
                        String[] line = scannerTarget.nextLine().trim().split(" "); //
Splitting the dataset

                        for (int j=0; j<line.length; j++) {
                            arrayTarget[i][j] = Integer.parseInt(line[j]); // Parsing
String to Integer and save to array
                        }
                    }
                }
        } catch (FileNotFoundException e)
        {
            e.printStackTrace();
        }
        return arrayTarget;
    }

private static PowerDatacenter datacenter1, datacenter2, datacenter3, datacenter4, datacenter5, datacenter6;
    /** The cloudlet list. */
private static List<Cloudlet> cloudletList;

    /** The vmList. */
private static List<Vm> vmList;

private static List<Vm> createVM(int userId, int vms) {

    //Creates a container to store VMs.
    //This list is passed to the broker later
    LinkedList<Vm> list = new LinkedList<Vm>();

    //VM Parameters
    long size = 10000; //Image size (MB)
    int[] ram = {512,1024,2048}; //VM memory (MB)
    int[] mips = {400,500,600}; //VM processing power (MIPS)
    long bw = 1000; //VM bandwidth
    int pesNumber = 1; //Number of cpus
    String vmm = "Xen"; //VMM name

    //create VMs
    Vm[] vm = new Vm[vms];

    for(int i=0;i<vms;i++){
        //For loop to create a VM with a time shared scheduling policy for
cloudlets:
        vm[i] = new Vm(i, userId, mips[i%3], pesNumber, ram[i%3], bw, size, vmm, new CloudletSchedulerSpaceShared());
        list.add(vm[i]);
    }

    return list;
}

```

```

    }

private static ArrayList<Double> getSeedValue(int cloudletcount){

    // Creating an arraylist to store Cloudlet Datasets
    ArrayList<Double> seed = new ArrayList<Double>();
    Log.println(System.getProperty("user.dir")+ "/dataset/SDSCDatasetANN.txt");

    try{
        // Opening and scanning the file
        File fobj = new File(System.getProperty("user.dir")+
"/dataset/SDSCDatasetANN.txt");

        java.util.Scanner readFile = new java.util.Scanner(fobj);

        while(readFile.hasNextLine() && cloudletcount>0)
        {
            // Adding the file to the arraylist
            seed.add(readFile.nextDouble());
            cloudletcount--;
        }
        readFile.close();

    } catch (FileNotFoundException e) {
        e.printStackTrace();
    }

    return seed;
}

private static List<Cloudlet> createCloudlet(int userId, int cloudlets){

    ArrayList<Double> randomSeed = getSeedValue(cloudlets);

    // Creates a container to store Cloudlets
    LinkedList<Cloudlet> list = new LinkedList<Cloudlet>();

    //Cloudlet parameters
    long length = 0; // Cloudlet length (MI) - 0 for SDSC
    //long length = 1000; // Cloudlet length (MI) - 1000 for Random Dataset
    long fileSize = 300; // Cloudlet file size (MB)
    long outputSize = 300; // Cloudlet file size (MB)
    int pesNumber = 1; // Cloudlet CPU needed to process
    UtilizationModel utilizationModel = new UtilizationModelFull();

    Cloudlet[] cloudlet = new Cloudlet[cloudlets];

    for(int i=0;i<cloudlets;i++){
        long finalLen = length +
Double.valueOf(randomSeed.get(i)).longValue();
        // Creating the cloudlet with all the parameter listed
        cloudlet[i] = new Cloudlet(i, finalLen, pesNumber, fileSize, outputSize, utilizationModel, utilizationModel,
utilizationModel);

        // setting the owner of these Cloudlets
        cloudlet[i].setUserId(userId);
    }
}

```

```

        list.add(cloudlet[i]);
    }

    return list;
}

/**
 * Creates main() to run this example
 */
public static void main(String[] args) {
    Log.println("Starting Cloud Simulation Example...");

    try {
        // First step: Initialize the CloudSim package. It should be called
        // before creating any entities.
        int num_user = 1; // Number of grid users
        Calendar calendar = Calendar.getInstance();
        boolean trace_flag = false; // Mean trace events
        int hostId=0; // Starting host ID
        int vmNumber = 54; // The number of VMs created
        int cloudletNumber = 1479; // The number of Tasks created

        // Initialize the CloudSim library
        CloudSim.init(num_user, calendar, trace_flag);

        //Second step: Create Data Centers
        //Datacenters are the resource providers in CloudSim. We need at least one of them to run a CloudSim simulation
        datacenter1 = createDatacenter("DataCenter_1", hostId);
        hostId = 3;
        datacenter r2 = createDatacenter("DataCenter_2", hostId);
        hostId = 6;
        datacenter3 = createDatacenter("DataCenter_3", hostId);
        hostId = 9;
        datacenter4 = createDatacenter("DataCenter_4", hostId);
        hostId = 12;
        datacenter5 = createDatacenter("DataCenter_5", hostId);
        hostId = 15;
        datacenter6 = createDatacenter("DataCenter_6", hostId);

        //Third step: Create Broker
        DatacenterBroker broker = createBroker();
        int brokerId = broker.getId();

        //Fourth step: Create VMs and Cloudlets
        vmList = createVM(brokerId,vmNumber); //Creating vms
        cloudletList = createCloudlet(brokerId,cloudletNumber); // Creating
cloudlets

        //Fifth step: Send VMs and Cloudlets to broker
        broker.submitVmList(vmList);
        broker.submitCloudletList(cloudletList);
    }
}

```

```

//Sixth step: Use ANN
BasicNetwork network = (BasicNetwork)EncogDirectoryPersistence.loadObject(new File("ANNscheduler-
SDSC.EG"));

LENGTH_RAW_DATA = Reading2DArrayFromFileLength();
TARGET_RAW_DATA = Reading2DArrayFromFileTarget();

// Creating a normalization rules
//NormalizedField input = new NormalizedField(NormalizationAction.Normalize, null, 50000, 10000, 1, 0); //for
Random Dataset
NormalizedField input = new NormalizedField(NormalizationAction.Normalize, null, 8790000, 0, 1, 0); //for SDSC
NormalizedField output = new
NormalizedField(NormalizationAction.Normalize, null, 10, 0, 1, 0);

// Doing normalization to the Input
for (int m=0; m<LENGTH_RAW_DATA.length; m++) {
    for (int n=0; n<9; n++) {
        LENGTH_RAW_DATA[m][n] =
input.normalize(LENGTH_RAW_DATA[m][n]);
    }
}

// Doing normalization to the Output
for (int m=0; m<TARGET_RAW_DATA.length; m++) {
    for (int n=0; n<9; n++) {
        TARGET_RAW_DATA[m][n] =
output.normalize(TARGET_RAW_DATA[m][n]);
    }
}

// Create data
MLDataSet trainingSet = new
BasicMLDataSet(LENGTH_RAW_DATA, TARGET_RAW_DATA);
int iterator = 0; //Iterator for the Cloudlet IDs
Long placeholderLong; //Placeholder to convert long to integer

// Testing the ANN
for(MLDataPair pair: trainingSet ) {
    final MLData outputData = network.compute(pair.getInput());
    System.out.println("");
    System.out.println("For Input:");
    for (int a=0 ; a<9; a++) {

System.out.print(Math.round(input.deNormalize(pair.getInput().getData(a))) + " ");
    }
    System.out.println("");
    System.out.println("Actual Result:");
    for (int b=0 ; b<9; b++) {

System.out.print(Math.round(output.deNormalize(outputData.getData(b))) + " ");
    }
    System.out.println("");
    System.out.println("Assignment:");
    for (int c=0 ; c<9; c++) {

```

```

        placeholderLong = new
Long(Math.round(output.deNormalize(outputData.getData(c))));
        int VMidOutput = placeholderLong.intValue();

System.out.print(cloudletList.get(iterator*9+c).getCloudletId() + " Assigned to ");
        System.out.print((VMidOutput + iterator*9)%54);
broker.bindCloudletToVm(cloudletList.get(iterator*9+c).getCloudletId(), (VMidOutput + iterator*9)%54);
        System.out.println("");
    }
    System.out.println("");
    iterator++;
}

// Seventh step: Starts the simulation
CloudSim.startSimulation();

// Final step: Print results when simulation is over
List<Cloudlet> newList = broker.getCloudletReceivedList();

CloudSim.stopSimulation();

printCloudletList(newList);

Log.println("Cloud Simulation Example finished!");
}
catch (Exception e)
{
    e.printStackTrace();
    Log.println("The simulation has been terminated due to an unexpected
error");
}
}

private static PowerDatacenter createDatacenter(String name, int hostId){

    // Here are the steps needed to create a PowerDatacenter:
    // 1. We need to create a list to store one or more machines
    List<PowerHost> hostList = new ArrayList<PowerHost>();

    // 2. A Machine contains one or more PEs or CPUs/Cores. Therefore, should
    // create a list to store these PEs before creating a Machine.
    List<Pe> peList1 = new ArrayList<Pe>();
    List<Pe> peList2 = new ArrayList<Pe>();
    List<Pe> peList3 = new ArrayList<Pe>();

    int mipsunused= 300; // Unused core, only 3 cores will be able to process
    Cloudlets for this simulation

    int mips1 = 400; // The MIPS Must be bigger than the VMs
    int mips2 = 500;
    int mips3 = 600;

```

```

// 3. Create PEs and add these into the list.
//for a quad-core machine, a list of 4 PEs is required:
peList1.add(new Pe(0, new PeProvisionerSimple(mips1))); // need to store Pe id and MIPS Rating, Must be bigger
than the VMs

peList1.add(new Pe(1, new PeProvisionerSimple(mips1)));
peList1.add(new Pe(2, new PeProvisionerSimple(mips1)));
peList1.add(new Pe(3, new PeProvisionerSimple(mipsunused)));
peList2.add(new Pe(4, new PeProvisionerSimple(mips2)));
peList2.add(new Pe(5, new PeProvisionerSimple(mips2)));
peList2.add(new Pe(6, new PeProvisionerSimple(mips2)));
peList2.add(new Pe(7, new PeProvisionerSimple(mipsunused)));
peList3.add(new Pe(8, new PeProvisionerSimple(mips3)));
peList3.add(new Pe(9, new PeProvisionerSimple(mips3)));
peList3.add(new Pe(10, new PeProvisionerSimple(mips3)));
peList3.add(new Pe(11, new PeProvisionerSimple(mipsunused)));

//4. Create Hosts with its id and list of PEs and add them to the list of machines
int ram = 128000 ; //Host memory (MB), Must be bigger than the VMs
long storage = 1000000; //Host storage (MB)
int bw = 10000; //Host bandwidth
int maxpower = 117; // Host Max Power
int staticPowerPercentage = 50; // Host Static Power Percentage

hostList.add(
    new PowerHostUtilizationHistory(
        hostId, new RamProvisionerSimple(ram),
        new BwProvisionerSimple(bw),
        storage,
        peList1,
        new VmSchedulerTimeShared(peList1),
        new PowerModelLinear(maxpower, staticPowerPercentage)));
hostId++;

hostList.add(
    new PowerHostUtilizationHistory(
        hostId, new RamProvisionerSimple(ram),
        new BwProvisionerSimple(bw),
        storage,
        peList2,
        new VmSchedulerTimeShared(peList2),
        new PowerModelLinear(maxpower, staticPowerPercentage)));
hostId++;

hostList.add(
    new PowerHostUtilizationHistory(
        hostId, new RamProvisionerSimple(ram),
        new BwProvisionerSimple(bw),
        storage,
        peList3,
        new VmSchedulerTimeShared(peList3),
        new PowerModelLinear(maxpower, staticPowerPercentage)));

```

```

// 5. Create a DatacenterCharacteristics object that stores the
// properties of a data center: architecture, OS, list of
// Machines, allocation policy: time- or space-shared, time zone
// and its price (G$/Pe time unit).
String arch = "x86";           // System architecture
String os = "Linux";           // Operating system
String vmm = "Xen";             // Name
double time_zone = 10.0;        // Time zone this resource located
double cost = 3.0;              // The cost of using processing in this resource
double costPerMem = 0.05;       // The cost of using memory in this
resource
double costPerStorage = 0.1;    // The cost of using storage in this resource
double costPerBw = 0.1;         // The cost of using bw in this
resource

LinkedList<Storage> storageList = new LinkedList<Storage>();

DatacenterCharacteristics characteristics = new DatacenterCharacteristics(
arch, os, vmm, hostList, time_zone, cost, costPerMem, costPerStorage,
costPerBw);

// 6. Finally, we need to create a PowerDatacenter object.
PowerDatacenter datacenter = null;
try {
datacenter = new PowerDatacenter(name, characteristics, new PowerVmAllocationPolicySimple(hostList),
storageList, 9);
} catch (Exception e) {
e.printStackTrace();
}

return datacenter;
}

private static DatacenterBroker createBroker(){

DatacenterBroker broker = null;
try {
broker = new DatacenterBroker("Broker");
} catch (Exception e) {
e.printStackTrace();
return null;
}
return broker;
}

/**
 * Prints the Cloudlet objects
 * @param list list of Cloudlets
 * @throws FileNotFoundException
 */
private static void printCloudletList(List<Cloudlet> list) throws FileNotFoundException {

```



```

        // Initializing the printed output to zero
        int size = list.size();
        Cloudlet cloudlet = null;

        String indent = "  ";
        Log.println();

        Log.println("===== OUTPUT =====");
        Log.println("Cloudlet ID" + indent + "STATUS" + indent +
            "Data center ID" + indent + "VM ID" + indent + "Time"
            + indent + "Start Time" + indent + "Finish Time" + indent +
            "Waiting Time");

        double waitTimeSum = 0.0;
        double CPUTimeSum = 0.0;
        int totalValues = 0;
        DecimalFormat dft = new DecimalFormat("###.##");

        double response_time[] = new double[size];

        // Printing all the status of the Cloudlets
        for (int i = 0; i < size; i++) {
            cloudlet = list.get(i);
            Log.print(cloudlet.getCloudletId() + indent + indent);

            if (cloudlet.getCloudletStatus() == Cloudlet.SUCCESS){
                Log.print("SUCCESS");
                CPUTimeSum = CPUTimeSum + cloudlet.getActualCPUTime();
                waitTimeSum = waitTimeSum + cloudlet.getWaitingTime();
                Log.println(indent + indent + indent + (cloudlet.getResourceId()-1) + indent + indent + indent +
                    cloudlet.getVmId() +
                    indent + indent + dft.format(cloudlet.getActualCPUTime()) + indent + indent +
                    dft.format(cloudlet.getExecStartTime())+
                    indent + indent + dft.format(cloudlet.getFinishTime())+ indent + indent + indent +
                    dft.format(cloudlet.getWaitingTime()));
                totalValues++;

                response_time[i] = cloudlet.getActualCPUTime();
            }
        }

        DoubleSummaryStatistics stats = DoubleStream.of(response_time).summaryStatistics();

        // Show the parameters and print them out
        Log.println();
        System.out.println("min = " + stats.getMin());
        System.out.println("Response_Time: " + CPUTimeSum/totalValues);

        Log.println();
        Log.println("TotalCPUTime : " + CPUTimeSum);
        Log.println("TotalWaitTime : " + waitTimeSum);
        Log.println("TotalCloudletsFinished : " + totalValues);
        Log.println();
        Log.println();

```

```

//Average Cloudlets Finished
Log.println("AverageCloudletsFinished : " + (CPUTimeSum/ totalValues));

//Average Start Time
double totalStartTime =0.0;
for (int i = 0; i < size; i++) {
    totalStartTime = cloudletList.get(i).getExecStartTime();
}
double avgStartTime = totalStartTime/size;
System.out.println("Average StartTime: " + avgStartTime );

//Average Execution Time
double ExecTime =0.0;
for (int i = 0; i < size; i++) {
    ExecTime = cloudletList.get(i).getActualCPUTime();
}
double avgExecTime = ExecTime/size;
System.out.println("Average Execution Time: " + avgExecTime );

//Average Finish Time
double totalTime =0.0;
for (int i = 0; i < size; i++) {
    totalTime = cloudletList.get(i).getFinishTime();
}
double avgTAT = totalTime/size;
System.out.println("Average FinishTime: " + avgTAT );

//Average Waiting Time
double avgWT = cloudlet.getWaitingTime()/size;
System.out.println("Average Waiting time: " + avgWT);

Log.println();
Log.println();

//Throughput
double maxFT =0.0;
for (int i = 0; i < size; i++) {
    double currentFT = cloudletList.get(i).getFinishTime();
    if (currentFT > maxFT) {
        maxFT = currentFT;
    }
}
double throughput = size/maxFT;
System.out.println("Throughput: " + throughput );

//Makespan
double makespan =0.0;
double makespan_total = makespan + cloudlet.getFinishTime();
System.out.println("Makespan: " + makespan_total);

//Imbalance Degree
double degree_of_imbalance = (stats.getMax() - stats.getMin())/(CPUTimeSum/ totalValues);
System.out.println("Imbalance Degree: " + degree_of_imbalance);

//Scheduling Length

```

```

double scheduling_length = waitTimeSum + makespan_total;
Log.println("Total Scheduling Length: " + scheduling_length);

//CPU Resource Utilization
double resource_utilization = (CPUTimeSum / (makespan_total * 54)) * 100;
Log.println("Resource Utilization: " + resource_utilization);

//Energy Consumption
Log.println(String.format("Total Energy Consumption: %.2f kWh",
(datacenter1.getPower() + datacenter2.getPower()+ datacenter3.getPower()+ datacenter4.getPower()+
datacenter5.getPower()+ datacenter6.getPower())/ (3600*1000)));
    }
}

```

C. Kode Sumber *GeneticAlgorithm.java*

```

package org.cloudbus.cloudsim.examples;

import java.util.List;

import org.cloudbus.cloudsim.Cloudlet;
import org.cloudbus.cloudsim.Vm;

/**
 * The GeneticAlgorithm class is our main abstraction for managing the
 * operations of the genetic algorithm. This class is meant to be
 * problem-specific, meaning that (for instance) the "calcFitness" method may
 * need to change from problem to problem.
 */
public class GeneticAlgorithm {
    private int populationSize;
    public static List<Cloudlet> cloudletList;
    public static List<Vm> vmList;

    /**
     * Mutation rate is the fractional probability than an individual gene will
     * mutate randomly in a given generation. The range is 0.0-1.0, but is
     * generally small (on the order of 0.1 or less).
     */
    private double mutationRate;

    /**
     * Crossover rate is the fractional probability that two individuals will
     * "mate" with each other, sharing genetic information, and creating
     * offspring with traits of each of the parents. Like mutation rate the
     * range is 0.0-1.0 but small.
     */
    private double crossoverRate;

    /**
     * Elitism is the concept that the strongest members of the population

```

```

* should be preserved from generation to generation. If an individual is
* one of the elite, it will not be mutated or crossover.
*/

    private int elitismCount;

@SuppressWarnings("static-access")
public GeneticAlgorithm(int populationSize, double mutationRate, double crossoverRate, int elitismCount,
List<Cloudlet> cloudletList, List<Vm> vmList) {
    this.populationSize = populationSize;
    this.mutationRate = mutationRate;
    this.crossoverRate = crossoverRate;
    this.elitismCount = elitismCount;
    this.cloudletList = cloudletList;
    this.vmlist = vmList;
}

/**
 * Initialize population
 */
* @param chromosomeLength
*     The length of the individual's chromosome
* @return population The initial population generated
*/
public Population initPopulation(int chromosomeLength, int dataCenterIterator) {
    // Initialize population
    Population population = new Population(this.populationSize, chromosomeLength,
dataCenterIterator);

    return population;
}

/**
 * Get Poisson Distribution for the chances of VM to fail
 */
* @param lambda
*     The fault rate at which a VM is going to fail
* @param x
*     x is going to 0, the number of VM we want to fail
* @return the chances of 0 VM to fail
*/
    static int factorial(int n)
    {
        if (n == 0)
            return 1;
        else
            return(n * factorial(n-1));
    }

public static int getRandomPoisson(double lambda)
{
    double L = Math.exp(-lambda);
    double p = 1.0;
    int k = 0;

    do
    {

```

```

        k++;
        p *= Math.random();
    } while (p > L);

    return k - 1;
}

public static double getPoisson(double lambda, int x, int n)
{
    double L = Math.exp(-lambda) * n;
    double p = Math.pow(lambda * n, x);
    int k = factorial(x);
    double result;

    result = L * p / k;

    return result;
}

/**
 * Calculate fitness for an individual.
 *
 * In this case, the fitness score is very simple: it's the number of ones
 * in the chromosome. Don't forget that this method, and this whole
 * GeneticAlgorithm class, is meant to solve the problem in the "AllOnesGA"
 * class and example. For different problems, you'll need to create a
 * different version of this method to appropriately calculate the fitness
 * of an individual.
 *
 * @param individual
 *       the individual to evaluate
 * @return double The fitness value for individual
 */
public double calcFitness(Individual individual, int dataCenterIterator, int cloudletIteration) {

    double totalExecutionTime = 0;
    double mips = 0;
    double failureRate = 0.04847468455;
    int iterator=0;
    dataCenterIterator = dataCenterIterator-1;

    for (int i=0 + dataCenterIterator*9 + cloudletIteration*54; i<9 +
dataCenterIterator*9 + cloudletIteration*54; i++)
    {
        int gene = individual.getGene(iterator);
        if (gene%9 == 0 || gene%9 == 3 || gene%9 == 6)
        {
            mips = 400;
        }else if (gene%9 == 1 || gene%9 == 4 || gene%9 == 7)
        {
            mips = 500;
        }else if (gene%9 == 2 || gene%9 == 5 || gene%9 == 8)
        {
            mips = 600;
        }else break;
    }
}

```

```

        //Log.println("Gene " + gene);
        totalExecutionTime = totalExecutionTime +
cloudletList.get(i).getCloudletLength() / mips;
        iterator++;
    }

    int random = getRandomPoisson(failureRate);
    double poisson=(getPoisson(failureRate, random, 9));

    // Calculate fitness
    double fitness = 0.90 * (1/totalExecutionTime) + 0.1 * (1/poisson);
    //Log.println("Fitness " + fitness);

    // Store fitness
    individual.setFitness(fitness);

    return fitness;
}

/**
 * Evaluate the whole population
 */
 * Essentially, loop over the individuals in the population, calculate the
 * fitness for each, and then calculate the entire population's fitness. The
 * population's fitness may or may not be important, but what is important
 * Here is making sure that each individual gets evaluated.
 */
 * @param population
 * the population to evaluate
 */
public void evalPopulation(Population population, int dataCenterIterator, int cloudletIteration) {

    // Loop over population evaluating individuals and summing population fitness
    double populationFitness=0;

    for (Individual individual : population.getIndividuals()) {

        double individualFitness = calcFitness(individual, dataCenterIterator,
cloudletIteration);

        individual.setFitness(individualFitness);
        populationFitness+=individualFitness;

    }
    population.setPopulationFitness(populationFitness);

}

/**
 * Check if population has met termination condition
 */
 * For this simple problem, we know what a perfect solution looks like, so
 * we can simply stop evolving once we've reached a fitness level.
 */
 * @param population

```

```

* @return boolean True if termination condition met, otherwise, false
*/

/**
 * Select parent for crossover
 *
 * @param population
 * The population to select parent from
 * @return The individual selected as a parent
 */
public Individual selectParent(Population population) {
    // Get individuals
    Individual individuals[] = population.getIndividuals();

    // Spin roulette wheel
    double populationFitness = population.getPopulationFitness();
    double rouletteWheelPosition = Math.random() * populationFitness;

    // Find parent
    double spinWheel = 0;
    for (Individual individual : individuals) {
        spinWheel += individual.getFitness();
        if (spinWheel >= rouletteWheelPosition) {
            return individual;
        }
    }
    return individuals[population.size() - 1];
}

/**
 * Apply crossover to population
 *
 * Crossover, more colloquially considered "mating", takes the population
 * and blends individuals to create new offspring. It is hoped that when two
 * individuals crossover that their offspring will have the strongest
 * qualities of each of the parents. Of course, it's possible that an
 * offspring will end up with the weakest qualities of each parent.
 *
 * This method considers both the GeneticAlgorithm instance's crossoverRate
 * and the elitismCount.
 *
 * The type of crossover we perform depends on the problem domain. We don't
 * want to create invalid solutions with crossover, so this method will need
 * to be changed for different types of problems.
 *
 * This particular crossover method selects random genes from each parent.
 *
 * @param population
 * The population to apply crossover to
 * @return The new population
 */
public Population crossoverPopulation(Population population, int dataCenterIterator) {
    // Create new population
    Population newPopulation = new Population(population.size());

```

```

        // Loop over current population by fitness
        for (int populationIndex = 0; populationIndex < population.size();
populationIndex++) {

            Individual parent1 = population.getFittest(populationIndex);

            // Apply crossover to this individual?
            if (this.crossoverRate > Math.random() && populationIndex >
this.elitismCount) {

                // Initialize offspring
                Individual offspring = new
Individual(parent1.getChromosomeLength(), dataCenterIterator);

                // Find second parent
                Individual parent2 = selectParent(population);

                // Loop over genome
                for (int geneIndex = 0; geneIndex <
parent1.getChromosomeLength(); geneIndex++) {

                    // Use half of parent1's genes and half of parent2's genes
                    if (0.5 > Math.random()) {
                        offspring.setGene(geneIndex,
parent1.getGene(geneIndex));
                    } else {
                        offspring.setGene(geneIndex,
parent2.getGene(geneIndex));
                    }
                }

                // Add offspring to new population
                newPopulation.setIndividual(populationIndex, offspring);
            } else {
                // Add individual to new population without applying crossover
                newPopulation.setIndividual(populationIndex, parent1);
            }
        }

        return newPopulation;
    }

    /**
     * Apply mutation to population
     */
    /**
     * Mutation affects individuals rather than the population. We look at each
     * individual in the population, and if they're lucky enough (or unlucky, as
     * it were), apply some randomness to their chromosome. Like crossover, the
     * The type of mutation applied depends on the specific problem we're solving.
     * In this case, we simply randomly flip 0s to 1s and vice versa.
     */
    /**
     * This method will consider the GeneticAlgorithm instance's mutationRate
     * and elitismCount
     */
    /**
     * @param population
     * The population to apply mutation to
     * @return The mutated population
     */

```



```

public Population mutatePopulation(Population population, int dataCenterIterator) {
    // Initialize new population
    Population newPopulation = new Population(this.populationSize);
    dataCenterIterator = dataCenterIterator - 1;

    // Loop over current population by fitness
    for (int populationIndex = 0; populationIndex < population.size();
populationIndex++) {

        Individual individual = population.getFittest(populationIndex);

        // Loop over individual's genes
        for (int geneIndex = 0; geneIndex < individual.getChromosomeLength();
geneIndex++) {

            // Skip mutation if this is an elite individual
            if (populationIndex > this.elitismCount) {
                // Does this gene need mutation?
                if (this.mutationRate > Math.random()) {
                    // Get new gene
                    int newGene=0 + 9 * dataCenterIterator;
                    if (individual.getGene(geneIndex)%9 == 0) {
                        double r=Math.random();
                        if(r<0.125)
                        {
                            newGene=1 + 9 * dataCenterIterator;
                        }
                        else if(r>0.125 && r<0.250)
                        {
                            newGene=2 + 9 * dataCenterIterator;
                        }
                        else if(r>0.250 && r<0.375)
                        {
                            newGene=3 + 9 * dataCenterIterator;
                        }
                        else if(r>0.375 && r<0.5)
                        {
                            newGene=4 + 9 * dataCenterIterator;
                        }
                        else if(r>0.5 && r<0.625)
                        {
                            newGene=5 + 9 * dataCenterIterator;
                        }
                        else if(r>0.625 && r<0.75)
                        {
                            newGene=6 + 9 * dataCenterIterator;
                        }
                        else if(r>0.75 && r<0.875)
                        {
                            newGene=7 + 9 * dataCenterIterator;
                        }
                        else
                        {
                            newGene=8 + 9 * dataCenterIterator;
                        }
                    }
                    else if (individual.getGene(geneIndex)%9 ==

```

1) {

```
        double r=Math.random();
        if(r<0.125)
        {
            newGene=0 + 9 * dataCenterIterator;
        }
        else if(r>0.125 && r<0.250)
        {
            newGene=2 + 9 * dataCenterIterator;
        }
        else if(r>0.250 && r<0.375)
        {
            newGene=3 + 9 * dataCenterIterator;
        }
        else if(r>0.375 && r<0.5)
        {
            newGene=4 + 9 * dataCenterIterator;
        }
        else if(r>0.5 && r<0.625)
        {
            newGene=5 + 9 * dataCenterIterator;
        }
        else if(r>0.625 && r<0.75)
        {
            newGene=6 + 9 * dataCenterIterator;
        }
        else if(r>0.75 && r<0.875)
        {
            newGene=7 + 9 * dataCenterIterator;
        }
        else
        {
            newGene=8 + 9 * dataCenterIterator;
        }
    }
    else if (individual.getGene(geneIndex)%9 ==
```

2) {

```
        double r=Math.random();
        if(r<0.125)
        {
            newGene=0 + 9 * dataCenterIterator;
        }
        else if(r>0.125 && r<0.250)
        {
            newGene=1 + 9 * dataCenterIterator;
        }
        else if(r>0.250 && r<0.375)
        {
            newGene=3 + 9 * dataCenterIterator;
        }
        else if(r>0.375 && r<0.5)
        {
            newGene=4 + 9 * dataCenterIterator;
        }
        else if(r>0.5 && r<0.625)
```

3) {

```
{
    newGene=5 + 9 * dataCenterIterator;
}
else if(r>0.625 && r<0.75)
{
    newGene=6 + 9 * dataCenterIterator;
}
else if(r>0.75 && r<0.875)
{
    newGene=7 + 9 * dataCenterIterator;
}
else
{
    newGene=8 + 9 * dataCenterIterator;
}
}
else if (individual.getGene(geneIndex)%9 ==
```

4) {

```
double r=Math.random();
```

5) {

```
        if(r<0.125)
        {
            newGene=0 + 9 * dataCenterIterator;
        }
        else if(r>0.125 && r<0.250)
        {
            newGene=1 + 9 * dataCenterIterator;
        }
        else if(r>0.250 && r<0.375)
        {
            newGene=2 + 9 * dataCenterIterator;
        }
        else if(r>0.375 && r<0.5)
        {
            newGene=3 + 9 * dataCenterIterator;
        }
        else if(r>0.5 && r<0.625)
        {
            newGene=5 + 9 * dataCenterIterator;
        }
        else if(r>0.625 && r<0.75)
        {
            newGene=6 + 9 * dataCenterIterator;
        }
        else if(r>0.75 && r<0.875)
        {
            newGene=7 + 9 * dataCenterIterator;
        }
        else
        {
            newGene=8 + 9 * dataCenterIterator;
        }
    }
    else if (individual.getGene(geneIndex)%9 ==

        double r=Math.random();
        if(r<0.125)
        {
            newGene=0 + 9 * dataCenterIterator;
        }
        else if(r>0.125 && r<0.250)
        {
            newGene=1 + 9 * dataCenterIterator;
        }
        else if(r>0.250 && r<0.375)
        {
            newGene=2 + 9 * dataCenterIterator;
        }
        else if(r>0.375 && r<0.5)
        {
            newGene=3 + 9 * dataCenterIterator;
        }
        else if(r>0.5 && r<0.625)
        {
            newGene=4 + 9 * dataCenterIterator;
```

6) {

```
    }  
    else if(r>0.625 && r<0.75)  
    {  
        newGene=6 + 9 * dataCenterIterator;  
    }  
    else if(r>0.75 && r<0.875)  
    {  
        newGene=7 + 9 * dataCenterIterator;  
    }  
    else  
    {  
        newGene=8 + 9 * dataCenterIterator;  
    }  
}  
else if (individual.getGene(geneIndex)%9 ==  
  
    double r=Math.random();  
    if(r<0.125)  
    {  
        newGene=0 + 9 * dataCenterIterator;  
    }  
    else if(r>0.125 && r<0.250)  
    {  
        newGene=1 + 9 * dataCenterIterator;  
    }  
    else if(r>0.250 && r<0.375)  
    {  
        newGene=2 + 9 * dataCenterIterator;  
    }  
    else if(r>0.375 && r<0.5)  
    {  
        newGene=3 + 9 * dataCenterIterator;  
    }  
    else if(r>0.5 && r<0.625)  
    {  
        newGene=4 + 9 * dataCenterIterator;  
    }  
    else if(r>0.625 && r<0.75)  
    {  
        newGene=5 + 9 * dataCenterIterator;  
    }  
    else if(r>0.75 && r<0.875)  
    {  
        newGene=7 + 9 * dataCenterIterator;  
    }  
    else  
    {  
        newGene=8 + 9 * dataCenterIterator;  
    }  
}  
else if (individual.getGene(geneIndex)%9 ==
```

7) {

```
    double r=Math.random();  
    if(r<0.125)  
    {
```

8) {

```
        newGene=0 + 9 * dataCenterIterator;
    }
    else if(r>0.125 && r<0.250)
    {
        newGene=1 + 9 * dataCenterIterator;
    }
    else if(r>0.250 && r<0.375)
    {
        newGene=2 + 9 * dataCenterIterator;
    }
    else if(r>0.375 && r<0.5)
    {
        newGene=3 + 9 * dataCenterIterator;
    }
    else if(r>0.5 && r<0.625)
    {
        newGene=4 + 9 * dataCenterIterator;
    }
    else if(r>0.625 && r<0.75)
    {
        newGene=5 + 9 * dataCenterIterator;
    }
    else if(r>0.75 && r<0.875)
    {
        newGene=6 + 9 * dataCenterIterator;
    }
    else
    {
        newGene=8 + 9 * dataCenterIterator;
    }
}
else if (individual.getGene(geneIndex)%9 ==

        double r=Math.random();
    if(r<0.125)
    {
        newGene=0 + 9 * dataCenterIterator;
    }
    else if(r>0.125 && r<0.250)
    {
        newGene=1 + 9 * dataCenterIterator;
    }
    else if(r>0.250 && r<0.375)
    {
        newGene=2 + 9 * dataCenterIterator;
    }
    else if(r>0.375 && r<0.5)
    {
        newGene=3 + 9 * dataCenterIterator;
    }
    else if(r>0.5 && r<0.625)
    {
        newGene=4 + 9 * dataCenterIterator;
    }
    else if(r>0.625 && r<0.75)
```

```

        {
            newGene=5 + 9 * dataCenterIterator;
        }
        else if(r>0.75 && r<0.875)
        {
            newGene=6 + 9 * dataCenterIterator;
        }
        else
        {
            newGene=7 + 9 * dataCenterIterator;
        }
    }
    // Mutate gene
    individual.setGene(geneIndex, newGene);
}

}

// Add individual to population
newPopulation.setIndividual(populationIndex, individual);
}

// Return mutated population
return newPopulation;
}

}

```

D. Kode Sumber *Population.Java*

```

package org.cloudbus.cloudsim.examples;

import java.util.Arrays;
import java.util.Comparator;
import java.util.Random;

/**
 * A population is an abstraction of a collection of individuals. The population
 * class is generally used to perform group-level operations on its individuals,
 * such as finding the strongest individuals, collecting stats on the population
 * as a whole, and selecting individuals to mutate or crossover.
 */
public class Population {
    public Individual population[];
    public double populationFitness = -1;

    /**
     * Initializes blank population of individuals
     *
     * @param populationSize
     */
}

```

```

*      The number of individuals in the population
*/
public Population(int populationSize) {
    // Initial population
    this.population = new Individual[populationSize];
}

/**
 * Initializes population of individuals
 *
 * @param populationSize
 *      The number of individuals in the population
 * @param chromosomeLength
 *      The size of each individual's chromosome
 */
public Population(int populationSize, int chromosomeLength, int dataCenterIterator) {
    // Initialize the population as an array of individuals
    this.population = new Individual[populationSize];

    // Create each individual in turn
    for (int individualCount = 0; individualCount < populationSize;
individualCount++) {
        // Create an individual, initializing its chromosome to the given length
        Individual individual = new Individual(chromosomeLength,
dataCenterIterator);

        // Add individual to population
        this.population[individualCount] = individual;
    }
}

/**
 * Get individuals from the population
 *
 * @return individuals Individuals in population
 */
public Individual[] getIndividuals() {
    return this.population;
}

/**
 * Find an individual in the population by its fitness
 *
 * This method lets you select an individual in order of its fitness. This
 * can be used to find the single strongest individual (eg, if you're
 * testing for a solution), but it can also be used to find weak individuals
 * (if you're looking to cull the population) or some of the strongest
 * individuals (if you're using "elitism").
 *
 * @param offset
 *      The offset of the individual you want, sorted by fitness. 0 is
 *      the strongest, population.length - 1 is the weakest.
 * @return individual Individual at offset
 */
public Individual getFittest(int offset) {
    // Order population by fitness

```



```

        Arrays.sort(this.population, new Comparator<Individual>() {
            @Override
            public int compare(Individual o1, Individual o2) {
                if (o1.getFitness() < o2.getFitness()) {
                    return 1;
                } else if (o1.getFitness() > o2.getFitness()) {
                    return -1;
                }
                return 0;
            }
        });

        // Return the fittest individual
        return this.population[offset];
    }

    /**
     * Set population's group fitness
     *
     * @param fitness
     * The population's total fitness
     */
    public void setPopulationFitness(double fitness) {
        this.populationFitness = fitness;
    }

    /**
     * Get population's group fitness
     *
     * @return populationFitness The population's total fitness
     */
    public double getPopulationFitness() {
        return this.populationFitness;
    }

    /**
     * Get population's size
     *
     * @return size The population's size
     */
    public int size() {
        return this.population.length;
    }

    /**
     * Set individual at offset
     *
     * @param individual
     * @param offset
     * @return individual
     */
    public Individual setIndividual(int offset, Individual individual) {
        return population[offset] = individual;
    }

```

```

        /**
         * Get individual at offset
         *
         * @param offset
         * @return individual
         */
    public Individual getIndividual(int offset) {
        return population[offset];
    }

    /**
     * Shuffles the population in-place
     *
     * @param void
     * @return void
     */
    public void shuffle() {
        Random rnd = new Random();
        for (int i = population.length - 1; i > 0; i--) {
            int index = rnd.nextInt(i + 1);
            Individual a = population[index];
            population[index] = population[i];
            population[i] = a;
        }
    }
}

```

E. Kode Sumber *Individual.java*

```

package org.cloudbus.cloudsim.examples;

/**
 * An "Individual" represents a single candidate solution. The core piece of
 * information about an individual is its "chromosome", which is an encoding of
 * a possible solution to the problem at hand. A chromosome can be a string, an
 * array, a list, etc -- in this class, the chromosome is an integer array.
 *
 * An individual position in the chromosome is called a gene, and these are the
 * atomic pieces of the solution that can be manipulated or mutated. When the
 * chromosome is a string, as in this case, each character or set of characters
 * can be a gene.
 *
 * An individual also has a "fitness" score; this is a number that represents
 * how good a solution to the problem this individual is. The meaning of the
 * fitness score will vary based on the problem at hand.
 */
public class Individual {
    public int[] chromosome;
    private double fitness = -1;

    /**
     * Initializes individual with specific chromosome
     */
}

```

```

        *
        * @param chromosome
        * The chromosome to give individual
        */
public Individual(int[] chromosome) {
    // Create individual chromosome
    this.chromosome = chromosome;
}

/**
 * Initializes random individuals.
 */
*
* This constructor assumes that the chromosome is made entirely of 0s and
* 1s, which may not always be the case, so make sure to modify as
* necessary. This constructor also assumes that a "random" chromosome means
* simply picking random zeroes and ones, which also may not be the case
* (for instance, in a traveling salesman problem, this would be an invalid
    * solution).
    *
    * @param chromosomeLength
    * The length of the individual's chromosome
    */
public Individual(int chromosomeLength, int dataCenterIterator) {

    this.chromosome = new int[chromosomeLength];
    dataCenterIterator = dataCenterIterator-1;

    int max = 8 + 9 * dataCenterIterator;
    int min = 0 + 9 * dataCenterIterator;
    int range = max - min + 1;

    // generate random numbers within 0 to 8
    for (int gene = 0; gene < chromosomeLength; gene++) {
        int rand = (int)(Math.random() * range) + min;
        this.setGene(gene, rand);
    }
}

/**
 * Gets individual's chromosome
 */
* @return The individual's chromosome
    */
    public int[] getChromosome() {
        return this.chromosome;
    }

    /**
    * Gets individual's chromosome length
    *
    * @return The individual's chromosome length
    */
    public int getChromosomeLength() {
        return this.chromosome.length;
    }

```

```

        /**
         * Set gene at offset
         *
         * @param gene
         * @param offset
         * @return gene
         */
public void setGene(int offset, int gene) {
    this.chromosome[offset] = gene;
}

        /**
         * Get gene at offset
         *
         * @param offset
         * @return gene
         */
public int getGene(int offset) {
    return this.chromosome[offset];
}

        /**
         * Store individual's fitness
         *
         * @param fitness
         * The individual's fitness
         */
public void setFitness(double fitness) {
    this.fitness = fitness;
}

        /**
         * Gets individual's fitness
         *
         * @return The individual's fitness
         */
public double getFitness() {
    return this.fitness;
}

        /**
         * Display the chromosome as a string.
         *
         * @return string representation of the chromosome
         */
public String toString() {
    String output = "";
    for (int gene = 0; gene < this.chromosome.length; gene++) {
        output += this.chromosome[gene];
    }
    return output;
}
}

```

F. Kode Sumber ANNTTest.java

```
package org.cloudbus.cloudsim.examples;

import java.io.BufferedReader;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.util.Scanner;

import org.encog.Encog;
import org.encog.engine.network.activation.ActivationReLU;
import org.encog.engine.network.activation.ActivationSigmoid;
import org.encog.ml.data.MLData;
import org.encog.ml.data.MLDataPair;
import org.encog.ml.data.MLDataSet;
import org.encog.ml.data.basic.BasicMLDataSet;
import org.encog.neural.networks.BasicNetwork;
import org.encog.neural.networks.layers.BasicLayer;
import org.encog.neural.networks.training.propagation.back.Backpropagation;
import org.encog.neural.networks.training.propagation.manhattan.ManhattanPropagation;
import org.encog.neural.networks.training.propagation.quick.QuickPropagation;
import org.encog.persist.EncogDirectoryPersistence;
import org.encog.util.arrayutil.NormalizationAction;
import org.encog.util.arrayutil.NormalizedField;

public class ANNTTest {

    // Double Array to hold the raw length data
    public static double LENGTH_RAW_DATA[][];

    // Double Array to hold the raw target data
    public static double TARGET_RAW_DATA[][];

    public static double[][] Reading2DArrayFromFileLength()
    {
        Scanner scannerLength;
        int rows = 653; // Number of rows to be scanned
        int columns = 9; // Number of columns to be scanned
        double [][] arrayLength = new double[rows][columns];

        try
        {
            scannerLength = new Scanner(new BufferedReader(new
            FileReader(System.getProperty("user.dir")+ "/train/DatasetLength-SDSC.txt")));
            while(scannerLength.hasNextLine()) {
                for (int i=0; i<arrayLength.length; i++) {
                    String[] line = scannerLength.nextLine().trim().split(" "); // Splitting
                    the dataset
                    for (int j=0; j<line.length; j++) {
                        arrayLength[i][j] = Integer.parseInt(line[j]); // Parsing String to
                        Integer and save to array
                    }
                }
            }
        }
    }
}
```

```

        }
    }
} catch (FileNotFoundException e)
{
    e.printStackTrace();
}
return arrayLength;
}

public static double[][] Reading2DArrayFromFileTarget()
{
    Scanner scannerTarget;
    int rows = 653; // Number of rows to be scanned
    int columns = 9; // Number of columns to be scanned
    double [][] arrayTarget = new double[rows][columns];

    try
    {
        scannerTarget = new Scanner(new BufferedReader(new
FileReader(System.getProperty("user.dir")+ "/train/DatasetTarget-SDSC.txt")));
        while(scannerTarget.hasNextLine()) {
            for (int i=0; i<arrayTarget.length; i++) {
                String[] line = scannerTarget.nextLine().trim().split(" "); // Splitting
the dataset

                for (int j=0; j<line.length; j++) {
                    arrayTarget[i][j] = Integer.parseInt(line[j]); // Parsing String to
Integer and save to array
                }
            }
        }
    } catch (FileNotFoundException e)
    {
        e.printStackTrace();
    }
    return arrayTarget;
}

/**
 * The main method.
 * @param args No arguments are used.
 */
public static void main(final String args[]) {

    // Saving the data scanned into the double arrays
    LENGTH_RAW_DATA = Reading2DArrayFromFileLength();
    TARGET_RAW_DATA = Reading2DArrayFromFileTarget();

    // Create a neural network
    BasicNetwork network = new BasicNetwork();
    network.addLayer(new BasicLayer(null,true,9)); // 9 input nodes
    network.addLayer(new BasicLayer(new ActivationReLU(),true,18)); // 18 hidden
nodes
    network.addLayer(new BasicLayer(new ActivationSigmoid(),false,9)); // 9 output
nodes

```

```

        network.getStructure().finalizeStructure();
        network.reset();

        // Creating a normalization rules
        //NormalizedField input = new NormalizedField(NormalizationAction.Normalize,
null, 50000, 10000, 1, 0); //for Random Dataset
        NormalizedField input = new NormalizedField(NormalizationAction.Normalize,
null, 8790000, 0, 1, 0); //for SDSC
        NormalizedField output = new NormalizedField(NormalizationAction.Normalize,
null, 10, 0, 1, 0);

        // Doing normalization to the Input
        for (int m=0; m<LENGTH_RAW_DATA.length; m++) {
            for (int n=0; n<9; n++) {
                LENGTH_RAW_DATA[m][n] =
input.normalize(LENGTH_RAW_DATA[m][n]);
            }
        }

        // Doing normalization to the Output
        for (int m=0; m<TARGET_RAW_DATA.length; m++) {
            for (int n=0; n<9; n++) {
                TARGET_RAW_DATA[m][n] =
output.normalize(TARGET_RAW_DATA[m][n]);
            }
        }

        // Create training data
        MLDataSet trainingSet = new BasicMLDataSet(LENGTH_RAW_DATA,
TARGET_RAW_DATA);

        // Train the neural network
        final ManhattanPropagation train = new ManhattanPropagation(network,
trainingSet, 0.00001);

        int epoch = 1;

        do {
            train.iteration();
            System.out.println("Epoch #" + epoch + " Error:" + train.getError());
            epoch++;
        } while(epoch<100000 && train.getError(>0.12); // Epoch until 100000 or error
below 12% (Best Fit for ANN)

        train.finishTraining();

        // Test the neural network
        System.out.println("Neural Network Results:");
        for(MLDataPair pair: trainingSet ) {
            final MLData outputData = network.compute(pair.getInput());
            System.out.println("");
            System.out.println("For Input:");
            for (int a=0 ; a<9; a++) {

                System.out.print(Math.round(input.deNormalize(pair.getInput().getData(a))) + " ");

            }
            System.out.println("");

```

```

        System.out.println("Actual Result:");
        for (int b=0 ; b<9; b++) {

System.out.print(Math.round(output.deNormalize(outputData.getData(b))) + " ");

        }
        System.out.println("");
        System.out.println("Ideal Result:");
        for (int c=0 ; c<9; c++) {

System.out.print(Math.round(output.deNormalize(pair.getIdeal().getData(c))) + " ");

        }
        System.out.println("");
        System.out.println("");
    }

    // Saving the neural network
    EncogDirectoryPersistence.saveObject(new File("ANNscheduler.EG"), network);
    Encog.getInstance().shutdown();
}
}

```

G. Dataset Cloudlet

Untuk *Dataset* yang digunakan sebagai Panjang *Cloudlet (Task)* di dalam Simulasi *Genetic Algorithm* bisa diakses pada <https://intip.in/TABryanDatasetCloudlet>

H. Output Genetic Algorithm

Untuk *Dataset* hasil *output* dari *Genetic Algorithm* sebelum dilakukan proses *Preprocessing* bisa diakses pada <https://intip.in/TABryanOutputGA>

I. Dataset Artificial Neural Network

Untuk *Dataset* yang digunakan sebagai Panjang *Cloudlet (Task)* di dalam Simulasi *Artificial Neural Network* bisa diakses pada <https://intip.in/TABryanDatasetANN>

J. Dataset Train Artificial Neural Network

Untuk *Dataset* yang digunakan sebagai pembelajaran untuk pembuatan model *Artificial Neural Network* bisa diakses pada <https://intip.in/TABryanDatasetTrain>

K. Dataset Test Artificial Neural Network

Untuk *Dataset* yang digunakan sebagai pengujian untuk pembuatan model *Artificial Neural Network* bisa diakses pada <https://intip.in/TABryanDatasetTest>

L. Model Artificial Neural Network

Untuk model *Artificial Neural Network* yang sudah dilakukan pembelajaran bisa diakses pada <https://intip.in/TABryanModelANN>

BIODATA PENULIS



Bryan Yehuda Mannuel lahir di Kota Surabaya pada 9 Maret 2001. Penulis menempuh Pendidikan mulai dari SD Kristen Petra 7 (2007-2013), SMP Kristen Petra 3 (2013-2016), SMA Kristen Petra 2 (2016-2019) dan kemudian menjadi mahasiswa pada Departemen Teknologi Informasi Institut Teknologi Sepuluh Nopember (ITS) dengan rumpun mata kuliah *Cloud Computing* (2019-2023).

Selama perkuliahan penulis aktif dalam beberapa organisasi seperti menjadi pengurus Himpunan Mahasiswa Teknologi Informasi (HMIT) ITS sebagai Staf Ahli Departemen Riset dan Teknologi (Ristik), pengurus *Society of Renewable Energy* (SRE) ITS sebagai Manager *IT Development*, pengurus *A Renewable Agent* (ARA) ITS sebagai Manager *IT Development*, pengurus INI LHO ITS! (ILITS) ITS sebagai Staf Ahli *IT Development*, pengurus Pembinaan Kerohanian Mahasiswa Baru Kristen (PKMBK) Persekutuan Mahasiswa Kristen (PMK) ITS sebagai Staf Publikasi, Dokumentasi, dan Dekorasi (PDD), pengurus *Explore IT* (EXPLOIT) ITS sebagai Staf Acara, dan pengurus Persekutuan Mahasiswa Kristen (PMK) ITS sebagai Staf Ahli Media dan Informasi (MEDFO).

Penulis juga pernah mengikuti beberapa kegiatan Merdeka Belajar seperti Kredensial Mikro Mahasiswa Indonesia (KMMI) pada tahun 2021 dan mendapatkan beasiswa dari program IISMA untuk bisa belajar selama satu semester di *Hanyang University*, Korea Selatan pada tahun 2022. Penulis juga perenah memenangkan juara satu pada perlombaan *Huawei ICT Competition* tahun 2020-2021 pada *Cloud Track*.

Penulis dapat dijangkau dan dihubungi lebih lanjut melalui surat elektronik pada bryanyehuda@gmail.com