OPTIMASI GIZI PADA ANAK PANTI ASUHAN DENGAN *IMMUNE GENETIC ALGORITHM*

SKRIPSI

Untuk memenuhi sebagian persyaratan
memperoleh gelar Sarjana Komputer

Disusun oleh:
Iqbal Dhea Furqon Tauhid
NIM: 125150200111055

PROGRAM STUDI TEKNIK INFORMATIKA
JURUSAN TEKNIK INFORMATIKA
FAKULTAS ILMU KOMPUTER
UNIVERSITAS BRAWIJAYA
MALANG
2016

repository.ub.ac.id
UNIVERSITAS BRAWIJAYA

PENGESAHAN

OPTIMASI GIZI PADA ANAK PANTI ASUHAN DENGAN *IMMUNE GENETIC ALGORITHM*

SKRIPSI

Diajukan untuk memenuhi sebagian persyaratan
memperoleh gelar Sarjana Komputer

Disusun Oleh :
Iqbal Dhea Furqon Tauhid
NIM: 125150200111055

Skripsi ini telah diuji dan dinyatakan lulus pada
15 Agustus 2016
Telah diperiksa dan disetujui oleh:

Dosen Pembimbing I	Dosen Pembimbing II
Imam Cholissodin, S.Si., M.Kom.	M. Tanzil Furqon, S.Kom., M.CompSc.
NIK: 201201 850719 1 001	NIP: 19820930 200801 1 004

Mengetahui
Ketua Jurusan Teknik Informatika

Tri Astoto Kurniawan, S.T, M.T, Ph.D
NIP: 19710518 200312 1 001

PERNYATAAN ORISINALITAS

Saya menyatakan dengan sebenar-benarnya bahwa sepanjang pengetahuan saya, di dalam naskah skripsi ini tidak terdapat karya ilmiah yang pernah diajukan oleh orang lain untuk memperoleh gelar akademik di suatu perguruan tinggi, dan tidak terdapat karya atau pendapat yang pernah ditulis atau diterbitkan oleh orang lain, kecuali yang secara tertulis disitasi dalam naskah ini dan disebutkan dalam daftar pustaka.

Apabila ternyata didalam naskah skripsi ini dapat dibuktikan terdapat unsur-unsur plagiasi, saya bersedia skripsi ini digugurkan dan gelar akademik yang telah saya peroleh (sarjana) dibatalkan, serta diproses sesuai dengan peraturan perundang-undangan yang berlaku (UU No. 20 Tahun 2003, Pasal 25 ayat 2 dan Pasal 70).

Malang, 21 Juli 2016

Iqbal Dhea Furqon Tauhid

NIM: 125150200111055

KATA PENGANTAR

Puji syukur kehadirat Allah SWT karena atas berkat dan karunia-Nya penulis dapat menyelesaikan skripsi yang berjudul "**Optimasi Gizi pada Anak Panti Asuhan dengan *Immune Genetic Algorithm***" untuk memenuhi sebagian persyaratan memperoleh gelar Sarjana Komputer.

Dalam penulisan skripsi ini tak lepas bantuan, dukungan, serta bimbingan dari berbagai pihak. Oleh karena itu ucapan terima kasih yang sebesar besarnya penulis sampaikan kepada,

1. Bapak Imam Cholissodin, S.Si., M.Kom. selaku dosen pembimbing I atas bimbingan serta waktu yang dicurahkan kepada penulis dalam penyelesaian skripsi ini.
2. Bapak M. Tanzil Furqon, S,Kom., M.CompSc. selaku dosem pembimbing II atas bimbingan serta waktu yang dicurahkan kepada penulis dalam penyelesaian skripsi ini.
3. Ibu Laksmi Karunia Tanuwijaya, S.Gz., M.Biomed. yang telah meluangkan waktunya untuk memberi pengetahuan serta bimbingan tentang ilmu gizi yang penulis butuhkan untuk menyelesaikan skripsi ini.
4. Dosen Program Teknologi Informasi dan Ilmu Komputer Universitas Brawijaya yang telah memberi bekal ilmu pengetahuan kepada penulis selama masa perkuliahan.
5. Orang tua tercinta Rudi Hartono dan Nanik Indahyati, serta saudara saudara tersayang yang tak pernah lelah memberi dukungan, motivasi, doa, dan segala pengorbanannya baik secara moril maupun materiil yang sangat berarti bagi penulis selama penyusunan skripsi ini.
6. Sahabat dan teman seperjuangan skripsi yang telah berbagi pengetahuan dan dukungan untuk penyelesaian skripsi ini, Cika Herdanis, Aldino Caturrahmanto dan Elyana Putri Budianti.
7. Teman dan sahabat penulis yang selalu memberikan semangat dalam penyusunan skripsi ini, Viqi Hanada, Sandhy Wistara, M. Faisal Amir, Aries Satryadhi Aldana, serta teman dan sahabat lainnya yang tak mungkin penulis sebutkan satu persatu.
8. Teman teman Program Teknologi Informasi dan Ilmu Komputer angkatan 2012 yang terlibat dalam penyusunan skripsi ini
9. Serta berbagai pihak lainnya yang tak dapat penulis sebutkan satu per satu, penulis ucapkan terima kasih atas segala bantuan yang telah diberikan.

Penulis sadar atas segala kekurangan yang ada saat penyusunan skripsi ini, oleh karena itu penulis harapkan saran serta kritik yang membangun demi pengembangan penelitian selanjutnya. Semoga skripsi ini dapat memberi manfaat bagi pembaca serta untuk pengembangan penelitian selanjutnya.

Malang, 21 Juni 2015

Penulis

Iqbal.dft@gmail.com

repository.ub.ac.id

ABSTRAK

Gizi merupakan zat yang dibutuhkan manusia bagi pertumbuhan dan kesehatannya. Meskipun begitu asupan gizi yang kurang tepat juga dapat mengakibatkan berbagai permasalahan yang mengganggu kesehatan. Sayangnya masih dapat dijumpai permasalahan gizi di berbagai tempat, salah satunya di panti asuhan. *Immune Genetic Algorithm* merupakan algoritma genetika yang dioptimasi dengan konsep imunitas dalam biologi dengan tujuan meningkatkan kemampuan algoritma genetika dalam pencarian optimum lokal sementara tetap menjaga kemampuannya dalam pencarian optimum global. Pencarian solusi rekomendasi menu makanan didapat dari mengkombinasikan kromosom yang diproses dengan operator genetika (*crossover* dan mutasi) yang kemudian dioptimasi dengan metode vaksinasi lalu diseleksi dengan proses seleksi imun. Hasil dari pengujian menunjukkan nilai *fitness* terbaik didapatkan dengan parameter: jumlah populasi awal = 70, probabilitas *crossover* = 0,7, probabilitas mutasi = 0,7. konstanta *affinity* = 0,8, konstanta *similarity* = 0,7, serta *death factor* = 0,7. Algoritma ini dapat memberikan rekomendasi menu makanan yang dapat memenuhi kebutuhan gizi tiap hari untuk satu populasi, hanya saja biaya dari menu makanan yang dihasilkan masih cukup tinggi.

Kata kunci: gizi, rekomendasi menu makanan, panti asuhan, algoritma genetika*, immune genetic algorithm,* seleksi imun

UNIVERSITAS BRAWIJAYA

repository.ub.ac.id

ABSTRACT

Nutrition is a subtance that human need for growth and health. However, lack of proper nutrition could lead to various problems that interfere with health. Unfortunately nutritional problem still could be found in various places, one of them is in orphanage. Immune Genetic Algorithm is genetic algorithm that optimized with the concept of immunity in biology to improving the ability of genetic algorithm in search of local optimum while maintain its ability in search of global optimum. The search of menu recommendation is obtained from combination of chromosome which processed with genetic operator (crossover and mutation) and then optimized with vaccination method and then selected by immune selection process. The result of testing showing the best fitness value obtained with parameters: initial population number = 70, crossover probability = 0.7, mutation probability = 0.7, affinity constant = 0.8, similarity constant = 0.7, and death factor = 0.7. This algorithm could provide food recommendation which can meet the daily nutritional needs for a population, however the cost the resulting menu is still quite high.

***Keywords**: nutrition, food recomendation, orphanage, genetic algorithm, immune genetic algorithm, immune selection*

UNIVERSITAS BRAWIJAYA

DAFTAR ISI

PENGESAHAN ii
PERNYATAAN ORISINALITAS iii
KATA PENGANTAR iv
ABSTRAK vi
ABSTRACT vii
DAFTAR ISI viii
DAFTAR TABEL xii
DAFTAR GAMBAR xiii
DAFTAR LAMPIRAN xv
BAB 1 PENDAHULUAN 1
1.1 Latar belakang 1
1.2 Rumusan masalah 3
1.3 Tujuan 3
1.4 Manfaat 3
1.5 Batasan masalah 3
1.6 Sistematika pembahasan 4
1.7 Jadwal pelaksanaan 4
BAB 2 LANDASAN KEPUSTAKAAN 6
2.1 Kajian pustaka 6
2.2 Gizi 10
2.2.1 Angka Kecukupan Gizi (AKG) 10
2.2.2 Karbohidrat 10
2.2.3 Lemak 11
2.2.4 Protein 11
2.3 Panti Asuhan 11
2.4 Algoritma Genetika 12
2.5 Artificial Immune System 13
2.6 Immune Genetic Algorithm 14
2.6.1 Representasi Kromosom 16
2.6.2 Inisialisasi 17

2.6.3 Ekstraksi Vaksin 17
2.6.4 Perhitungan *fitness* 17
2.6.5 Crossover 18
2.6.6 Mutasi 18
2.6.7 Vaksinasi 19
2.6.8 Seleksi Imun 19
2.6.9 Operasi *Update* 21
2.6.10 Kondisi Berhenti 21
BAB 3 METODOLOGI PENELITIAN 22
3.1 Studi Literatur 22
3.2 Analisis Kebutuhan 23
3.2.1 Kebutuhan fungsional 23
3.2.2 Kebutuhan non fungsional 23
3.3 Pengumpulan Data 23
3.4 Perancangan Sistem 24
3.5 Implementasi Sistem 25
3.6 Pengujian Algoritma 25
3.6.1 Pengujian jumlah populasi awal 25
3.6.2 Pengujian probabilitas *crossover* 26
3.6.3 Pengujian probabilitas mutasi 27
3.6.4 Pengujian konstanta *affinity* 27
3.6.5 Pengujian konstanta *similarity* 28
3.6.6 Pengujian konstanta *death factor* 29
3.7 Penarikan Kesimpulan 29
BAB 4 PERANCANGAN 30
4.1 Formulasi permasalahan 30
4.2 Siklus *Immune Genetic Algorithm* 30
4.2.1 Perhitungan kebutuhan gizi 32
4.2.2 Generasi populasi awal 33
4.2.3 Perhitungan nilai *fitness* 35
4.2.4 Ekstraksi Vaksin 39
4.2.5 Proses *crossover* 44

4.2.6 Proses *Mutasi* 46
4.2.7 Vaksinasi 48
4.2.8 Seleksi Imun 50
4.3 Perancangan Antarmuka 56
4.3.1 Antarmuka Halaman Awal 56
4.3.2 Antarmuka Detail Hasil 57
BAB 5 IMPLEMENTASI SISTEM 58
5.1 Implementasi *Immune Genetic Algorithm* 58
5.1.1 Implementasi perhitungan kebutuhan gizi 58
5.1.2 Implementasi generasi populasi awal 59
5.1.3 Implementasi perhitungan nilai *fitness* 60
5.1.4 Implementasi ekstraksi vaksin 64
5.1.5 Implementasi proses *crossover* 68
5.1.6 Implementasi proses mutasi 70
5.1.7 Implementasi vaksinasi 71
5.1.8 Implementasi seleksi imun 72
5.2 Implementasi antarmuka 76
5.2.1 Antarmuka Halaman Awal 76
5.2.2 Antarmuka Detail Hasil 77
BAB 6 PENGUJIAN DAN ANALISIS 78
6.1 Pengujian jumlah populasi awal 78
6.2 Pengujian probabilitas *crossover* 80
6.3 Pengujian probabilitas mutasi 82
6.4 Pengujian konstanta *affinity* 84
6.5 Pengujian konstanta *similarity* 87
6.6 Pengujian konstanta *death factor* 89
6.7 Analisa global 91
BAB 7 PENUTUP 93
7.1 Kesimpulan 93
7.2 Saran 94
DAFTAR PUSTAKA 95
LAMPIRAN A PERHITUNGAN MANUAL 97

LAMPIRAN B HASIL WAWANCARA..... 115

DAFTAR TABEL

Tabel 1.1 Jadwal pelaksanaan 5
Tabel 2.1 Kajian pustaka 8
Tabel 3.1 Pengujian jumlah populasi awal 26
Tabel 3.2 Pengujian probabilitas *crossover* 26
Tabel 3.3 Pengujian probabilitas mutasi 27
Tabel 3.4 Pengujian konstanta *affinity* 28
Tabel 3.5 Pengujian konstanta *similarity* 28
Tabel 3.6 Pengujian konstanta *death factor* 29
Tabel 6.1 Hasil pengujian populasi awal 78
Tabel 6.2 Hasil pengujian probabilitas *crossover* 81
Tabel 6.3 Hasil pengujian probabilitas mutasi 83
Tabel 6.4 Hasil pengujian konstanta *affinity* 85
Tabel 6.5 Hasil pengujian konstanta *similarity* 87
Tabel 6.6 Hasil pengujian konstanta *death factor* 89
Tabel 6.7 Contoh hasil menu makanan dari solusi terbaik 91
Tabel 6.8 Perkiraan gizi & biaya dari menu panti asuhan 92

DAFTAR GAMBAR

Gambar 2.1 *Pseudocode* algoritma genetika 12
Gambar 2.2 *Flowchart immune genetic algorithm* 15
Gambar 2.3 *Flowchart new immune genetic algorithm* 16
Gambar 2.4 Contoh representasi kromosom 17
Gambar 2.5 Contoh proses *crossover* 18
Gambar 2.6 Contoh proses mutasi 18
Gambar 2.7 Contoh proses vaksinasi vaksin lengkap 19
Gambar 2.8 Contoh proses vaksinasi vaksin sebagian 19
Gambar 3.1 Tahapan metodologi penelitian 22
Gambar 3.2 Model perancangan sistem 24
Gambar 4.1 Siklus *Immune Genetic Algorithm* 31
Gambar 4.2 Representasi kromosom 33
Gambar 4.3 Contoh kromosom 34
Gambar 4.4 Diagram alir generasi populasi awal 34
Gambar 4.5 Diagram alir perhitungan nilai *fitness* 38
Gambar 4.6 Diagram alir perhitungan variasi kromosom 39
Gambar 4.7 Diagram alir ekstraksi vaksin 40
Gambar 4.8 Ilustrasi pemilihan fragmen M1 41
Gambar 4.9 Ilustrasi pasangan M1 41
Gambar 4.10 Ilustrasi vaksin M1 41
Gambar 4.11 diagram alir ekstraksi vaksin M1 41
Gambar 4.12 Ilustrasi mencari nilai B_l 42
Gambar 4.13 Ilustrasi pasangan M2 42
Gambar 4.14 Ilustrasi vaksin M2 42
Gambar 4.15 Diagram alir ekstraksi vaksin M2 43
Gambar 4.16 Diagram alir penyimpanan vaksin 44
Gambar 4.17 *One point crossover* 44
Gambar 4.18 Diagram alir *crossover* 45
Gambar 4.19 Pemilihan antibodi untuk *crossover* 46
Gambar 4.20 *Uniform Mutation* 47

Gambar 4.21 Diagram alir proses mutasi 47
Gambar 4.22 Diagram alir pemilihan kandidat mutasi 48
Gambar 4.23 Vaksinasi 48
Gambar 4.24 Diagram alir proses vaksinasi 49
Gambar 4.25 Diagram alir proses seleksi imun 51
Gambar 4.26 Diagram alir proses hitung *affinity* 52
Gambar 4.27 Diagram alir proses hitung konsentasi 53
Gambar 4.28 Diagram alir proses perhitungan probabilitas seleksi 54
Gambar 4.29 Diagram alir proses seleksi antibodi 55
Gambar 4.31 Rancangan antarmuka halaman awal 56
Gambar 4.32 Rancangan antarmuka detail hasil 57
Gambar 5.1 Antarmuka halaman awal 76
Gambar 5.2 Antarmuka detail hasil 77
Gambar 6.1 Grafik *fitness* hasil pengujian jumlah populasi awal 79
Gambar 6.2 Grafik konvergensi hasil pengujian jumlah populasi awal 79
Gambar 6.3 Grafik *fitness* hasil pengujian probabilitas *crossover* 81
Gambar 6.4 Grafik konvergensi hasil pengujian probabilitas *crossover* 82
Gambar 6.5 Grafik *fitness* hasil pengujian probabilitas mutasi 83
Gambar 6.6 Grafik konvergensi hasil pengujian probabilitas mutasi 84
Gambar 6.7 Grafik *fitness* hasil pengujian konstanta *affinity* 86
Gambar 6.8 Grafik konvergensi hasil pengujian konstanta *affinity* 86
Gambar 6.9 Grafik pengujian *fitness* konstanta *similarity* 88
Gambar 6.10 Grafik pengujian konvergensi konstanta *similarity* 88
Gambar 6.11 Grafik pengujian *fitness* konstanta *death factor* 90
Gambar 6.12 Grafik pengujian konvergensi konstanta *death factor* 90

DAFTAR LAMPIRAN

LAMPIRAN A PERHITUNGAN MANUAL 97
A.1 Angka Kecukupan Gizi (AKG) 97
A.2 Data anak panti asuhan 97
A.3 Daftar makanan 98
A.4 Inisialisasi 100
A.5 Perhitungan nilai *fitness* 100
A.6 Ekstraksi Vaksin 105
A.7 *Crossover* 107
A.8 Mutasi 108
A.9 Vaksinasi 108
A.10 Perhitungan nilai *fitness* 109
A.11 Ekstraksi imun 109
A.12 Seleksi Imun 110
A.13 Akhir iterasi 113
LAMPIRAN B HASIL WAWANCARA 115
B.1 Wawancara di panti asuhan 115
B.2 Wawancara dengan ahli gizi 116

BAB 1 PENDAHULUAN

Bagian pendahuluan akan menjelaskan tentang latar belakang mengapa penelitian optimasi gizi pada anak panti asuhan ini perlu untuk dilakukan, selain itu juga menerangkan tentang rumusan masalah apa yang mendasari penelitian ini, tujuan apa yang ingin dicapai, manfaat penelitian ini dalam ruang lingkup yang lebih luas, batasan masalah yang akan membatasi ruang lingkup penelitian, serta sistematika pembahasan yang berisi struktur penulisan penelitian ini.

1.1 Latar belakang

Gizi merupakan zat yang dibutuhkan manusia bagi pertumbuhan dan kesehatannya, meskipun begitu asupan gizi yang kurang tepat juga dapat mengakibatkan berbagai permasalahan yang mengganggu kesehatan. Pada masa pertumbuhan, permasalahan gizi ini apabila tidak segera ditangani akibatnya dapat berimbas hingga dewasa (William, 2010) yang dapat berpengaruh pada kualitas sumber daya manusia. Apabila hal ini terjadi secara luas, maka dapat berpengaruh pada berkurangnya generasi muda (*loss generation*) hingga berdampak pada keadaan ekonomi bangsa (*loss economic*) di masa yang akan datang (Fauzi, 2012).

Permasalahan gizi pada masa pertumbuhan secara umum dapat digolongkan menjadi dua jenis, yaitu kekurangan gizi dan kelebihan gizi (Fauzi, 2012). Berdasarkan data dari Riset Kesehatan Dasar 2013 Provinsi Jawa Timur (Laksmiarti et al., 2013) pada rentang usia 5 hingga 12 tahun berdasarkan perbandingan indeks massa tubuh dengan umur (IMT/U), prevalensi sangat kurus dan obesitas di Jawa Timur berada di angka 3.5% dan 8.4%, sedangkan di kota Malang sendiri berada di angka 2.0% dan 13.2%. Pada rentang usia 13 hingga 15 tahun, prevalensi sangat kurus dan obesitas di Jawa Timur berturut turut 2.6% dan 3.0%, sedangkan di kota Malang sendiri berturut turut 3.0% dan 2.8%. Dari data tersebut, terlihat kota Malang sendiri belum dapat terbebas dari permasalahan gizi, baik gizi kurang maupun gizi berlebih.

Data diatas merupakan data yang diambil dari statistik populasi secara umum. Berdasarkan penelitian oleh (William, 2010) pada salah satu panti asuhan di Medan, dijumpai adanya penghuni yang menderita malnutrisi ringan sebesar 14.4%, dan *overweight* sebanyak 8.7%. Hal ini lebih mempertegas hasil temuan dari penelitian yang dilakukan oleh *Save the Children* dan Kementrian Sosial pada tahun 2007 yang dikutip oleh (Indonesia, 2011), yang salah satu hasil temuannya menunjukkan bahwa kurangnya pengetahuan pengurus panti tentang situasi anak serta pengasuhan yang seharusnya diterima anak. Selain itu hasil penelitian tersebut juga menggambarkan tentang buruknya kualitas makanan yang diterima oleh anak anak panti asuhan.

Dari hasil penelitian tersebut, Kementrian Sosial akhirnya menerbitkan Peraturan Mentri Sosial No. 3 tentang standar nasional pengasuhan anak untuk lembaga kesejahteraan sosial anak, yang salah satu poinnya ialah lembaga

diharuskan menjaga kualitas gizi dan nutrisi anak sesuai usia dan tumbuh kembang anak selama tinggal dalam lembaga kesejahteraan sosial anak (Indonesia, 2011).

Pada penelitian ini dilakukan studi kasus ke panti asuhan Nurul Abyadh yang beralamatkan pada Jl. Bendungan Sigura-gura 1 no. 8, Malang. Menurut keterangan pengurus panti asuhan tersebut, dalam satu bulan menghabiskan biaya sekitar 3 hingga 5 juta rupiah atau sekitar 1 juta rupiah per minggu untuk konsumsi anak anak panti asuhan yang berjumlah sekitar 40an anak. Menu makanan yang dikonsumsi biasanya mengacu pada menu yang telah ditetapkan untuk satu minggu kedepan dengan acuan perkiraan gizi yang hanya berdasarkan perkiraan pengurus panti tanpa perhitungan gizi yang jelas dalam penentuan menu tersebut. Saat makan tidak ada batasan porsi makan tiap anak, dengan kata lain tiap anak panti asuhan bebas untuk mengambil porsi makanan sesuka hati.

Banyaknya anak dalam panti asuhan serta terbatasnya dana untuk konsumsi yang berasal dari donatur tentunya menjadi pertimbangan utama dalam penentuan menu dan banyaknya makanan yang akan disediakan sehingga tetap dapat memenuhi kebutuhan gizi penghuninya. Di sisi lain, perencanaan menu makanan yang mengandung gizi optimal merupakan tugas yang gampang-gampang-susah, terbukti dengan masih dijumpainya kasus ketidakseimbangan gizi di masyarakat, terutapa di lingkungan panti asuhan seperti yang ditemukan oleh (William, 2010).

Salah satu cara yang dapat dilakukan untuk mempermudah pengurus panti asuhan dalam menentukan menu makanan yang optimal baik sesuai dengan kebutuhan gizi penghuni panti asuhan, ketersediaan dana yang ada, dan banyaknya makanan yang dibutuhkan, ialah dengan menerapkan sistem Optimasi Gizi pada Anak Panti Asuhan dengan *Immune Genetic Algorithm*.

Immune Genetic Algorithm merupakan algoritma genetika yang dioptimasi dengan konsep imunitas dalam biologi dengan tujuan meningkatkan kemampuan algoritma genetika dalam pencarian optimum lokal sementara tetap menjaga kemampuan algoritma genetika dalam pencarian optimum global (Wang et al., 1998). Penggunaan *Immune Genetic Algorithm* dalam penelitian ini didasarkan pada penelitian yang dilakukan oleh (Juan et al., 2009) yang berjudul "*Research on Application of IGA (Immune Genetic Algorithm) to the Solution of Course-Timetabling Problem*", penelitian menunjukkan bahwa *Immune Genetic Algorithm* dapat mengurangi fenomena degenerasi atau turunnya kualitas keturunan, serta dapat meningkatkan kecepatan konvergensi dalam setiap iterasi algoritma genetika. Senada dengan penelitian tersebut, (Jin & Fan, 2013) dengan judulnya "*A New Immune Genetic Algorithm for 0-1 Knapsack Problem*" juga memberi kesimpulan yang sama.

Dengan dilakukannya penelitian ini, diharapkan hasil dari penelitian ini dapat membantu untuk diterapkannya pola makan bergizi seimbang, yakni dengan

memberikan beberapa opsi contoh menu makanan yang dapat digunakan sehari hari.

1.2 Rumusan masalah

Berdasarkan latar belakang yang telah dijabarkan sebelumnya, dapat ditarik rumusan masalah sebagai berikut :

1. Bagaimana menerapkan *Immune Genetic Algorithm* untuk mengoptimasi gizi pada anak panti asuhan?
2. Bagaimana pengaruh parameter *Immune Genetic Algorithm* terhadap hasil optimasi gizi untuk anak panti asuhan?

1.3 Tujuan

Berdasarkan latar belakang dan rumusan masalah yang telah dijabarkan sebelumnya dapat disimpulkan tujuan dari penelitian ini ialah :

1. Menerapkan *Immune Genetic Algorithm* untuk mengoptimasi gizi untuk anak panti asuhan.
2. Mengetahui pengaruh parameter *Immune Genetic Algorithm* terhadap hasil optimasi gizi untuk anak panti asuhan.

1.4 Manfaat

Manfaat yang diharapkan dari penelitian ini ialah:

1. Bagi Penulis
   Dapat menerapkan ilmu yang didapat semasa perkuliahan, khususnya dalam implementasi algoritma evolusi.
2. Bagi Panti Asuhan
   Dapat membantu pengurus panti untuk memilih menu makanan yang paling optimal sesuai dengan kadar gizi dan biayanya.
3. Bagi Fakultas dan Universitas
   Dapat menambah jumlah penelitian khususnya dalam bidang algoritma evolusi, komputasi cerdas, dan ilmu komputer.

1.5 Batasan masalah

Dari uraian latar belakang, permasalahan, tujuan, dan manfaat yang diharapkan dari penelitian ini, untuk membatasi penelitian sehingga tidak melenceng terlalu jauh dari tujuan awal maka diberikan batasan masalah sebagai berikut :

1. Algoritma yang digunakan untuk optimasi ialah *Immune Genetic Algorithm*.
2. Tidak melakukan perbandingan dengan metode lain.
3. Optimasi gizi berpatokan pada kebutuhan energi anak sesuai jenis kelamin dan rentang usia tertentu sesuai dengan tabel Angka Kecukupan Gizi.
4. Data anak didapatkan dari panti asuhan Nurul Abyadh.
5. Harga yang digunakan adalah harga umum pada kota Malang.

1.6 Sistematika pembahasan

Untuk menjaga struktur penelitian agar lebih mudah dipahami, maka disusunlah sistematika pembahasan sebagai berikut:

1. Bab 1 Pendahuluan
   Bagian pendahuluan akan menjelaskan tentang latar belakang mengapa penelitian ini perlu untuk dilakukan, selain itu juga menerangkan tentang rumusan masalah apa yang mendasari penelitian ini, tujuan apa yang ingin dicapai, manfaat penelitian ini dalam ruang lingkup yang lebih luas, batasan masalah yang akan membatasi ruang lingkup penelitian, serta sistematika pembahasan yang berisi struktur penulisan penelitian ini.
2. Bab 2 Landasan Kepustakaan
   Landasan kepustakan akan menjelaskan tentang kajian pustaka dan dasar teori. Kajian pustaka menjelaskan tentang penelitian terdahulu. Dasar teori menjelaskan tentang teori yang diperlukan untuk penelitian berikutnya.
3. Bab 3 Metodologi Penelitian
   Metodologi akan menjelaskan metode penelitian yang akan digunakan dalam membuat sistem optimasi pola makanan berdasarkan gizi dengan *immune genetic algorithm* pada panti asuhan di kota Malang.
4. Bab 4 Perancangan
   Bab perancangan menjelaskan formulasi permasalahan, siklus *Immune Genetic Algorithm*, siklus penyelesaian optimasi gizi pada anak panti asuhan dengan *Immune Genetic Algorithm*, perhitungan manual, serta perancangan antar muka
5. Bab 5 Implementasi Sistem
   Implementasi sistem akan membahas tentang bagaimana sistem optimasi pola makanan berdasarkan gizi dengan *immune genetic algorithm* pada panti asuhan di kota Malang diimplementasikan.
6. Bab 6 Pengujian dan Analisis
   Pengujian dan analisis akan membahas tentang pengujian serta analisis hasil dari sistem.
7. Bab 7 Penutup
   Penutup akan menjelaskan tentang kesimpulan yang diperoleh dari pembuatan dan pengujian sistem optimasi pola gizi pada anak panti asuhan dengan *immune genetic algorithm*.

1.7 Jadwal pelaksanaan

Penelitian ini direncanakan akan dilaksanakan selama empat bulan. Rincian jadwal pelaksanaan penelitian ini dapat dilihat pada Tabel 1.1.

Tabel 1.1 Jadwal pelaksanaan

Keterangan	Bulan Ke-			
	1	2	3	4
Studi Literatur				
Analisis kebutuhan				
Pengumpulan data				
Perancangan				
Implementasi				
Pengujian				
Penarikan kesimpulan				
Pembuatan Laporan dan Dokumentasi				

BAB 2 LANDASAN KEPUSTAKAAN

Landasan kepustakan akan menjelaskan tentang kajian pustaka dan dasar teori. Kajian pustaka menjelaskan tentang penelitian terdahulu. Dasar teori menjelaskan tentang teori yang diperlukan untuk penelitian berikutnya.

2.1 Kajian pustaka

Penelitian pertama berjudul Implementasi Algoritma Genetika pada Optimasi Pemenuhan Kebutuhan Gizi yang dilakukan oleh (Pratiwi et al., 2014). Penelitian ini menggunakan algoritma genetika untuk mengoptimasi bahan makanan agar sesuai dengan kebutuhan gizinya dengan mempertimbangkan harga tiap bahan makanan. Perhitungan nilai *fitness*-nya yaitu dengan menggunakan selisih dari total harga dari 10 bahan makanan tertinggi dengan total harga bahan makanan pada tiap kromosom dan penalti. Perhitungan penalti dengan menghitung selisih total kandungan gizi dengan kebutuhan gizi jika total kandungan gizi tiap kromosom lebih rendah dari kebutuhan gizi. Penalti dihitung untuk masing-masing kebutuhan gizi, yaitu kalori, protein, lemak, dan karbohidrat. Untuk mendapatkan berapa banyak nutrisi yang diperlukan, penelitian ini menggunakan angka metabolisme basal yang dikalikan dengan aktifitas fisik untuk mencari nilai *total energy expenditure*, yang nantinya digunakan dalam perhitungan masing-masing kebutuhan nutrisi. Proses *crossover* dengan menggunakan metode *one point crossover*, yaitu dengan memilih satu titik persilangan secara acak, lalu menukar antar segmen kromosom induk untuk menghasilkan kromosom anak. Proses mutasi dilakukan dengan memilih beberapa gen pada kromosom secara acak, lalu mengubah isi gen tersebut secara acak sehingga terbentuk kromosom baru. Metode seleksi yang digunakan yaitu metode seleksi *elitism*, dimana hanya individu-individu yang memiliki nilai fitnes tertinggi yang dipertahankan untuk generasi selanjutnya. Representasi kromosom yang digunakan memiliki panjang bervariasi dengan jangkauan 5 hingga 15 yang mengakibatkan *offspring* yang dihasilkan lebih bervariasi dan dapat mempengaruhi nilai *fitness*. Hasil dari uji coba penelitian ini menunjukkan bahwa jumlah populasi dan jumlah generasi sangat mempengaruhi hasil, populasi optimal didapatkan pada nilai 40, sedangkan generasi optimal didapatkan pada nilai 500. Selain itu nilai probabiltias *crossover* dan mutasi didapatkan nilai optimal pada masing-masing 0,4 dan 0,6.

Penelitian kedua dilakukan oleh (Sari et al., 2014) yang berjudul Optimasi Asupan Gizi pada Ibu Hamil dengan menggunakan Algoritma Genetika. Penelitian yang dilakukan oleh Sari dkk ini juga menggunakan algoritma genetika, tetapi dengan objek yang sedikit berbeda yaitu optimasi asupan gizi untuk ibu hamil. Hampir sama dengan penelitian pertama, penelitian ini juga memperhitungkan gizi yang meliputi energi, protein, lemak, dan karbohidrat serta perhitungan biaya tiap bahan makanan. Perhitungan kebutuhan gizi juga memiliki beberapa kesamaan, hanya saja selain menggunakan angka metabolisme basal dan aktifitas fisik, dalam perhitungan *total energi expenditure* juga

memperhitungkan usia kehamilan, jam tidur, serta *specific dynamic action*. Perhitungan nilai *fitness* sama dengan penelitian pertama, yaitu total harga 10 bahan makanan tertinggi dikurangi dengan total harga bahan makanan tiap kromosom dan penalti. Metode *crossover* yang digunakan ialah *one point crossover*, sedangkan metode mutasi yang digunakan ialah *repiprocal exchange mutation* dimana metode mutasi ini memilih dua titik mutasi secara acak kemudian menukarkan nilai dari kedua titik tersebut. Seleksi elitis digunakan sebagai metode seleksinya, dimana pemilihan individu untuk generasi selanjutnya dipilih berdasarkan nilai *fitness* tertinggi. Representasi kromosom yang digunakan memiliki panjang antara 5 hingga 15 gen. Banyak generasi yang dibutuhkan untuk mencapai nilai optimum ialah 1500 generasi. Banyak populasi optimum mencapai nilai 150 individu. Kombinasi nilai probabilitas *crossover* dan mutasi yang paling optimum berada pada nilai 0,4 dan 0,6. Meskipun telah diperoleh nilai nilai optimum, optimasi yang didapatkan masih menunjukkan angka yang kurang dari kebutuhan energi ibu hamil.

Penelitian ketiga memiliki judul Optimasi Biaya Pemenuhan Gizi dan Nutrisi pada Manusia Lanjut Usia menggunakan Algoritma Genetika yang dilakukan oleh (Suci et al., 2015). Senada dengan dua penelitian sebelumnya, algoritma genetika dipilih sebagai metode untuk optimasi biaya pemenuhan gizi dan nutrisi pada manusia lanjut usia. Kandungan gizi yang digunakan juga meliputi energi, protein, lemak dan karbohidrat. Perhitungan nilai *fitness* juga memiliki kesamaan dengan dua penelitian sebelumnya. Metode *one point crossover* dipilih sebagai metode kawin silang antara dua kromosom, sedangkan *reciprocal exchange mutation* dipilih sebagai metode mutasinya. Seleksi elitis juga dipilih sebagai metode seleksi. Representasi kromosom yang digunakan memiliki panjang 14 kromosom yang masing-masing gen sesuai dengan menu seimbang untuk lanjut usia. Solusi optimal diperoleh pada ukuran populasi 120, generasi ke 750, serta probabilitas *crossover* dan mutasi pada 0,3 dan 0,7.

Penelitian keempat yang dilakukan oleh (Juan et al., 2009) berjudul *Research on Application of IGA (Immune Genetic Algorithm) to the Solution of Course-Timetabling Problem*. Penelitian ini menggunakan algoritma genetika yang telah dioptimasi dengan menambahkan operasi imun untuk menyelesaikan permasalahan penjadwalan. Kromosom direpresentasikan dengan gen berupa data dasar jadwal, seperti pengajar, pelajaran, jam pelajaran, serta ruang kelas. Metode seleksi yang dipilih oleh Juan dkk ialah selain menggunakan metode *Roulette Wheel Selection*, juga memperhitungkan hasil dari *immune operator*. *Immune operator* memperhitungkan seberapa besar probabilitas sebuah antibodi untuk dapat bertahan hingga generasi selanjutnya dengan mempertimbangkan *affinity* antar antibodi dan *affinity* dari antibodi dan antigen. Selain itu operasi imun ini juga menetapkan berapa jumlah antibodi atau individu generasi selanjutnya. *Immune memory* akan menyimpan antibodi yang memiliki nilai *fitness* atau *affinity* terhadap antigen terbaik dari generasi tersebut untuk menggantikan antibodi yang lebih buruk yang tersimpan dalam *memory*. Metode *crossover* dan mutasi yang digunakan sama dengan metode dari algoritma genetika, dengan sedikit penyesuaian dengan topik penjadwalan, yaitu dengan

one point crossover dan *uniform mutation*. Hasil yang didapat dari penelitian ini memperlihatkan bahwa *immune genetic algorithm* memiliki kelebihan daripada algoritma genetika biasa, algoritma ini dapat mencegah fenomena degenerasi yang terjadi pada algoritma genetika biasa sehingga dapat meningkatkan kecepatan konvergensi.

Penelitian kelima memiliki judul *A New Immune Genetic Algorithm for 0-1 Knapsack Problem* yang telah dilakukan oleh (Jin & Fan, 2013). Penelitian ini menggunakan *immune genetic algorithm* yang telah dioptimasi dengan menambahkan operasi vaksinasi setelah operasi *crossover* dan mutasi. Vaksinasi dilakukan dengan cara memilih antibodi dalam satu populasi untuk divaksinasi, vaksin diambil dari memori imun yang juga dipilih secara random. Beberapa gen dari antibodi terpilih akan dirubah sesuai dengan gen dari vaksin. Jika hasil dari vaksinasi memiliki nilai *fitness* yang lebih tinggi, maka akan menggantikan antibodi terpilih, sebaliknya jika nilai *fitness* lebih rendah maka antibodi terpilih tidak akan dirubah dan tetap masuk dalam populasi. Hasil dari penelitian ini menunjukkan selain dapat meningkatkan konvergensi, algoritma ini juga mencegah fenomena degradasi dalam setiap proses iterasi pada algoritma genetika serta menghasilkan nilai optimum yang lebih tinggi daripada algoritma genetika imun.

Ringkasan kajian pustaka dapat dilihat pada Tabel 2.1 berikut.

Tabel 2.1 Kajian pustaka

No.	Judul	Metode	Hasil
1.	Implementasi Algoritma Genetika pada Optimasi Pemenuhan Kebutuhan Gizi (Pratiwi et al., 2014)	Algoritma Genetika Proses : 1. Inisialisasi parameter awal 2. Membuat populasi baru 1. *Crossover* (*one point crossover)* 3. Mutasi 4. Perhitungan *fitness* 5. Seleksi (Seleksi elitis) 6. Pengecekan kondisi, jika belum optimum, kembali ke langkah 2 7. Jika sudah optimum tampilkan kromosom individu terbaik	Populasi optimal sesuai dengan uji coba ialah 40, sedangkan generasi optimal adalah 500. Peluang *crossover* dan mutasi yang paling optimal pada nilai 0,4 dan 0,6.
2.	Optimasi Asupan Gizi pada Ibu Hamil dengan menggunakan Algoritma Genetika (Sari et al., 2014)	Algoritma Genetika Proses : 2. Inisialisasi parameter awal 3. Membuat populasi awal 4. *Crossover* (*one point crossover)* 5. Mutasi (*repiprocal exchange mutation*) 6. Perhitungan *fitness* 7. Seleksi kromosom (Seleksi elitis) 8. Populasi baru	Generasi optimum didapatkan pada generasi ke 1500. Populasi optimum pada 150. Probabilitas *crossover* terbaik pada 0,4. Probabilitas mutasi terbaik pada 0,6. Nilai *fitness* dengan parameter tersebut masih kurang dari kebutuhan energi yang dibutuhkan ibu hamil.

Tabel 2.1 Kajian pustaka **(Lanjutan)**

No.	Judul	Metode	Hasil
		9. Pengecekan kondisi, jika belum optimum kembali ke langkah 3 10. Jika sudah optimum tampilkan kromosom terbaik	
3.	Optimasi Biaya Pemenuhan Gizi dan Nutrisi pada Manusia Lanjut Usia menggunakan Algoritma Genetika (Suci et al., 2015)	Algoritma Genetika Proses : 1. Inisialisasi parameter 2. Membuat populasi awal 3. *Crossover* 4. Mutasi (*repiprocal exchange mutation*) 5. Seleksi elitism 6. Populasi baru 7. Pengecekan kondisi 8. Jika sudah optimum tampilkan kromosom terbaik, jika belum kembali ke langkah 3	Generasi optimal pada nilai 500. Populasi optimal pada nilai 120. Kombinasi probabilitas *crossover* dan mutasi optimum pada 0,3 dan 0,7.
4.	*Research on Application of IGA (Immune Genetic Algorithm) to the Solution of Course-Timetabling Problem* (Juan et al., 2009)	*Immune Genetic Algorithm* Proses : 1. Inisialisasi parameter awal 2. Membuat populasi awal 3. Seleksi imun 4. Menyimpan antidbodi terbaik ke dalam memori imun 5. *Crossover* dan mutasi 6. Populasi baru 7. Pengecekan kondisi berhenti, jika masih belum optimum kembali ke langkah 3 8. Jika sudah optimum algoritma berhenti.	Penggunaan *Immune Genetic Algorithm* (IGA) dalam permasalahan penjadwalan menunjukkan bahwa IGA tak hanya cocok tapi juga efektifuntuk mencegah fenomena degenerasi yang biasa terjadi pada algoritma genetika biasa, selain itu IGA juga meningkatkan kecepatan konvergensi.
5.	A New *Immune Genetic Algorithm* for 0-1 Knapsack Problem (Jin & Fan, 2013)	*New Immunge Genetic Algorithm* Proses : 1. Inisialisasi awal 2. Ekstraksi vaksin 3. Perhitungan nilai *fitness* 4. Pengecekan kondisi berhenti, jika sudah optimal algoritma berhenti 5. *Crossover* dan mutasi 6. *Proses vaksinasi* 7. Seleksi imun 8. *Update* operation, (kembali ke langkah 2)	Algoritma *New Immune Genetic Algorithm* (NIGA) dapat meningkatkan konvergensi, serta mencegah fenomena degradasi dalam setiap proses iterasi pada algoritma genetika. Algoritma ini dapat menghasilkan nilai optimum yang lebih tinggi daripada *Immunge Genetic Algorithm* biasa.

Sumber : (Pratiwi et al., 2014), (Sari et al., 2014), (Suci et al., 2015), (Juan et al., 2009), (Jin & Fan, 2013)

2.2 Gizi

Istilah gizi mulai dikenal pada tahun 1950-an di Ind*one*sia. Kata gizi yang merupakan terjemahan dari kata dalam bahasa Inggris "*nutrition*" berasal dari bahasa arab "*ghidza*" yang dalam dialek mesir dibaca "*gizi*", sementara kata *nutrition* sendiri diterjemahkan langsung menjadi kata nutrisi yang memiliki arti serupa. Zat gizi diperlukan tubuh untuk memperoleh energi yang dapat digunakan untuk menjaga kelangsungan fungsi tubuh, tumbuh dan berkembang, serta melakukan aktifitas. Zat-zat gizi yang merupakan sumber energi utama bagi tubuh ialah karbohidrat, lemak, dan protein. (Muchtadi, 2014).

2.2.1 Angka Kecukupan Gizi (AKG)

Menurut Peraturan Menteri Kesehatan nomor 75 tahun 2013, Angka Kecukupan Gizi (AKG) merupakan suatu kecukupan rata rata zat gizi setiap hari bagi semua orang menurut golongan umur, jenis kelamin, ukuran tubuh, serta aktifitas tubuh untuk mencapai derajat kesehatan yang optimal. Fungsi dari AKG itu sendiri dapat digunakan dalam penilaian kadar kecukupan gizi, acuan penyusunan makanan sehari-hari, acuan perhitungan perencanaan penyediaan pangan tingkat regional maupun nasional, acuan pendidikan gizi, dan acuan label pangan yang mencantumkan informasi nilai gizi. (Indonesia, 2013).

2.2.2 Karbohidrat

Karbohidrat merupakan senyawa yang berfungsi sebagai sumber energi untuk tubuh. Karbohidrat merupakan sumber energi yang paling murah serta paling penting bagi makhluk hidup karena menyediakan unsur karbon yang siap digunakan oleh sel (dalam hal ini karbohidrat dalam bentuk pati, gula, atau glikogen). Menurut ukuran molekulnya, karbohidrat dibagi menjadi tiga kelompok, yaitu (Muchtadi, 2014) :

1. Monosakarida, karbohidrat yang paling sederhana yang terdiri dari satu unit *polihidroksi aldehid* atau keton. Karena rasa manisnya, monosakarida juga disebut sebagai gula sederhana. Contoh dari golongan ini ialah glukosa, fruktosa, dan galaktosa.
2. Oligosakarida, merupakan karbohidrat yang mengandung dua hingga sepuluh molekul gula sederhana (monosakarida) yang digabungkan dengan ikatan glikosida. Karbohidrat golongan disakarida merupakan oligosakarida yang paling banyak terdapat dalam bahan makananl, yaitu sukrosa, maltosa, dan laktosa.
3. Polisakarida, ialah karbohidrat yang memiliki molekul yang kompleks hingga ribuan molekul monosakarida. Polisakarida dibagi menjadi dua jenis, yaitu yang dapat dicerna (misal : pati, dekstrin, glikogen), dan yang tidak dapat dicerna (misal: selulosa, hemiselulosa, gum, pektin).

2.2.3 Lemak

Lemak memiliki peranan sebagai sember energi bagi tubuh. Lemak merupakan bentuk energi berlebih yang disimpan oleh hewan, dengan kata lain setiap bahan makanan hewani mengandung lemak. Lemak yang terdapat dalam bahan pangan nabati berbentuk minyak, terdapat pada serealia seperti jagung atau kacang-kacangan. Sebagian besar sayuran dan buah-buahan tidak mengandung lemak. Lemak juga memiliki peran sebagai pelarut dan pembawa vitamin larut lemak (vitamin A, D, E, dan K). Selain itu lemak juga dapat meningkatkan rasa enak atau kelezatan makanan (Muchtadi, 2014).

2.2.4 Protein

Protein merupakan senyawa yang terdapat pada setiap sel makhluk hidup. Selain sebagai sumber energi, protein juga bertindak sebagai zat pembangun dan zat pengatur tubuh. Beberapa fungsi protein yaitu (Muchtadi, 2014) :

1. Sumber energi
2. Pertumbuhan dan pemeliharaan jaringan
3. Pembentukan senyawa tubuh
4. Regulasi keseimbangan air
5. Mempertahankan netralitas tubuh
6. Pembentukan antibodi
7. Transportasi zat gizi

2.3 Panti Asuhan

Panti asuhan atau yang juga disebut dengan Lembaga Kesejahteraan Sosial Anak merupakan lembaga-lembaga kesejahteraan sosial yang dibentuk oleh pemerintah maupun masyarakat yang melaksanakan pengasuhan anak. Lembaga ini berperan dalam mendukung pengasuhan anak oleh keluarga serta memberi pelayanan bagi anak yang membutuhkan pengasuhan alternatif. Penempatan anak di lembaga seperti ini merupakan pilihan terakhir dari pengasuhan alternatif untuk anak yang tidak dapat diasuh dalam keluarga inti, keluarga besar, kerabat, maupun keluarga pengganti (Indonesia, 2011).

Hasil penelitian yang dilakukan oleh (William, 2010) pada panti asuhan di Medan menunjukkan bahwa masih terdapat kasus status gizi yang kurang baik di panti asuhan dengan persentasi 14,4 % malnutrisi ringan, serta 8,7 % *overweight*. Pada penelitian yang dilakukan oleh *Save the Children* bersama dengan Kementrian Sosial pada tahun 2007 yang dikutip oleh (Indonesia, 2011), salah satu hasil temuannya menunjukkan kurangnya pengetahuan pengurus panti tentang situasi anak serta pengasuhan yang seharusnya diterima oleh anak. Selain itu penelitian tersebut juga menggambarkan tentang buruknya kualitas makanan yang diterima oleh anak panti asuhan.

2.4 Algoritma Genetika

Algoritma genetika merupakan suatu teknik strategi adaptif dan optimisasi global. Tujuan utama dari algoritma genetika ialah untuk memaksimalkan hasil dari kandidat solusi dalam suatu populasi terhadap fungsi biaya dari domain problem. Strategi utama algoritma ini yaitu dengan berulangkali mengganti kombinasi solusi dengan metode rekombinasi dan mutasi pada mekanisme genetis dari suatu populasi kandidat solusi dengan mempertimbangkan fungsi biaya atau fungsi *fitness*. Secara umum algoritma ini diawali dengan inisialisasi populasi awal, kemudian evaluasi populasi tersebut untuk mendapatkan individu terbaik dalam satu populasi. Lakukan perulangan hingga mencapai kondisi selesai yang diinginkan untuk mendapatkan individu-individu yang lebih baik dengan menggunakan metode *crossover* dan mutasi (Brownlee, 2011).

Pseudocode dari algoritma genetika secara umum dapat dilihat pada Gambar 2.1 berikut.

```
Input : Population_size, Problem_size, P_crossover, P_mutation
Output : S_best
1  Population ← InitializePopulation (Population_size , Problem_size)
2  EvaluatePopulation(Population)
3  S_best ← GetBestSolution(Population)
4  While ¬StopCondition() do
5      Parents ← SelectParents(Population, Population_size)
6      Children ← ∅
7      Foreach Parent_1, Parent_2 ∈ Parent do
8          Child_1, Child_2 ← Crossover(Parent_1, Parent_2, P_crossover)
9          Children ← Mutate(Child_1, P_mutation)
10         Children ← Mutate(Child_2, P_mutation)
11     End
12     EvaluatePopulation(Children)
13     S_best ← GetBestSolution(Children)
14     Population ← Replace(Population, Children)
15 End
16 Return S_best
```

Gambar 2.1 *Pseudocode* algoritma genetika

Sumber : (Brownlee, 2011)

Berikut istiliah-istilah dasar yang sering digunakan dalam algoritma genetika (Sutojo et al., 2011),

1. **Gen** (*Genotype*) adalah variabel dasar yang membentuk suatu kromosom. Dalam algoritma genetika, gen ini dapat bernilai biner, *floating point*, *integer*, maupun karakter.
2. **Alel** (*Allele*) adalah nilai dari suatu gen. Sesuai dengan gennya, dapat berupa biner, *floating point*, *integer*, maupun karakter.
3. **Kromosom** (*Chromosome*) ialah susunan dari gen gen yang membentuk arti tertentu.
4. **Individu** merupakan kumpulan gen yang menyatakan salah satu kemungkinan solusi dari suatu permasalahan
5. **Populasi** ialah sekumpulan individu yang akan diproses secara bersama-sama dalam satu siklus proses evolusi
6. **Generasi** menyatakan satu satuan siklus proses evolusi
7. **Nilai *fitness*** menyatakan seberapa baik nilai dari suatu individu atau solusi yang didapatkan. Nilai *fitness* inilah yang digunakan sebagai acuan untuk mencapai nilai optimal

Algoritma yang terinspirasi dari teori Darwin ini telah diterapkan di berbagai bidang karena kekuatan pencarian paralel serta kemampuan pencarian globalnya. Hanya saja algoritma genetika klasik memiliki banyak keterbatasan, seperti kecenderungannya untuk mendapatkan hasil prematur, dan lambatnya kecepatan konvergensinya (Jin & Fan, 2013) (Juan et al., 2009).

2.5 Artificial Immune System

Artificial Immune System (AIS) merupakan sistem komputasi yang terinspirasi dari prinsip-prinsip dan proses dari sistem imunitas vertebrata. Algoritma ini memanfaatkan karakteristik pembelajaran (*learning)* dan ingatan (*memory*) hingga meniru tingkah laku dari sistem imun untuk menyelesaikan permasalahan (Yang, 2011). (Fang et al., 2012) menyebutkan jika algoritma ini lebih disarankan untuk aplikasi di bidang keamanan komputer mengingat sistem imun pada dasarnya merupakan sistem pertahanan terhadap patogen penyebab penyakit.

Secara umum, *artificial immune system* diawali dengan inisialisasi awal yaitu generasi populasi awal. Pada populasi awal tersebut dilakukan proses kalkulasi *affinity*, yaitu perhitungan kedekatan antara antibodi dan antigen serta dengan sesama antibodi, apabila terdapat antibodi yang sama persis maka antibodi tersebut harus dihapus dan digantikan dengan yang baru. Setelah itu dilakukan operator *clone*, dimana dipilih beberapa antibodi untuk dikloning atau digandakan kemudian dilakukan operasi *crossover* dan mutasi. Hasil dari proses kloning ini berupa antibodi baru, sehingga perlu dihitung ulang nilai *affinity*-nya untuk proses seleksi. Proses seleksi akan membandingkan antara antibodi anak (dari proses *cloning*) dengan induknya, jika *affinity* dari antibodi yang anak lebih baik daripada *affinity* antibodi induknya, maka antibodi tersebut akan menggantikan induknya. Langkah dari perhitungan *affinity* hingga seleksi akan

terus diulang hingga mencapai batas berhenti yang telah ditetapkan (Yang, 2011).

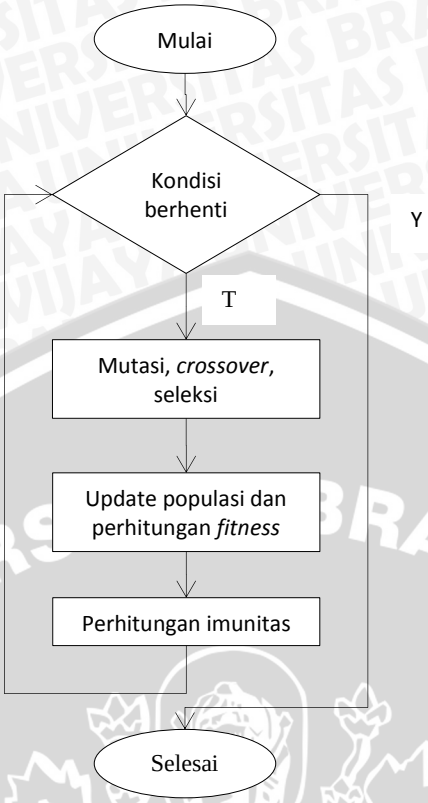
Istilah istilah yang sering digunakan dalam AIS ialah sebagai berikut (Weckman et al., 2012) (Bondal, 2008),

1. **Antibodi**, dalam konsep teori imunologi, antibodi merujuk pada sel dalam tubuh manusia yang mempunyai tugas untuk mempertahankan tubuh dari serangan infeksi atau virus yang menyerangnya. Dalam konteks algoritma ini, antibodi merupakan solusi potensial yang dihasilkan algoritma.
2. **Gen**, ialah bagian terkecil dari antibodi.
3. **Antigen**, merupakan infeksi yang menyerang tubuh, yang harus dielminasi oleh antibodi. Dalam konteks algoritma ini, antigen merupakan representasi masalah yang akan dicari solusinya.

2.6 Immune Genetic Algorithm

Algoritma genetika merupakan algoritma yang sangat baik digunakan untuk mencari nilai optimum global. Hanya saja pada titik tertentu sekitar 90 persen dari nilai *fitness* yang paling optimal terjadi fenomena osilasi yang memperlambat kecepatan untuk mencapai konvergensi serta menunjukkan berkurangnya kemampuan algoritma genetika dalam pencarian optimum lokal. *Immune Genetic Algorithm* (IGA) merupakan algoritma genetika yang dioptimasi dengan konsep imunitas dalam biologi dengan tujuan meningkatkan kemampuan algoritma genetika dalam pencarian optimum lokal sementara tetap menjaga kemampuan algoritma genetika dalam pencarian optimum global (Wang et al., 1998). Dengan kata lain, IGA merupakan gabungan antara algoritma genetika dengan algoritma *artificial immune system*. Pada penelitian yang lain menunjukkan bahwa IGA memiliki kecepatan konvergensi yang lebih baik daripada algoritma genetika standar dan dapat mencegah fenomena degenerasi yang mengakibatkan konvergensi prematur (Juan et al., 2009) (Jin & Fan, 2013).

Langkah langkah dasar dalam IGA mencakup inisialisasi awal, proses seleksi, *crossover* dan mutasi, perhitungan *fitness*, operator imun, serta pengecekan kondisi berhenti. *Flowchart* dasar dari algoritma ini dapat dilihat pada Gambar 2.2.

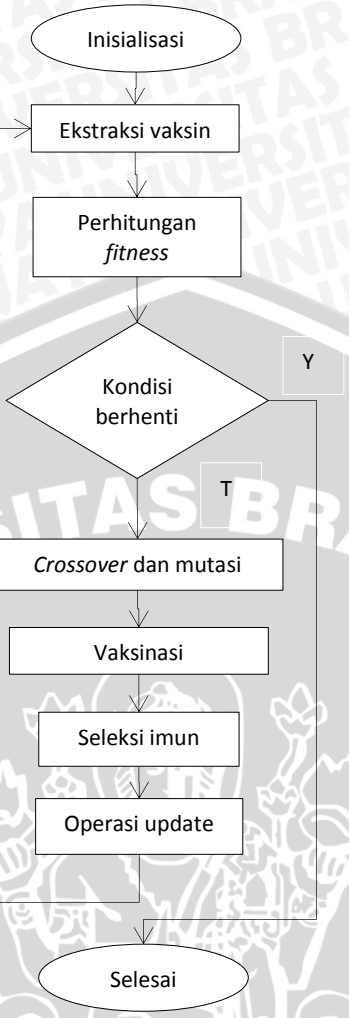


Gambar 2.2 *Flowchart immune genetic algorithm*

Sumber : (Wang et al., 1998)

Pada penelitian yang dilakukan oleh (Juan et al., 2009) menyimpan individu individu terbaik dalam generasi ke dalam memori imun dengan seleksi elitis sehingga terpisah dengan individu yang mengalami proses *crossover*, mutasi, dan operasi imun yang nantinya diseleksi dengan metode *roulette wheel selection*.

Pada penelitian yang lain IGA ditambahkan dengan operasi vaksinasi untuk meningkatkan probabilitas dalam mendapatkan nilai *fitness* yang lebih tinggi (Lei & Licheng, 2000) serta mencegah jatuhnya populasi dalam konvergensi lokal (Yan-cong et al., 2011). Lebih lanjut, (Jin & Fan, 2013) mengoptimisasi metode ekstraksi vaksin dengan menambah informasi lokasi vaksin tersebut untuk lebih mempercepat konvergensi dan mencegah konvergensi prematur, sehingga flowchart algoritma tersebut dapat dilihat pada Gambar 2.3.



Gambar 2.3 *Flowchart new immune genetic algorithm*

Sumber : (Jin & Fan, 2013)

Masing-masing langkah dari algoritma ini akan dijelaskan pada masing-masing subbab berikut :

2.6.1 Representasi Kromosom

Kromosom merupakan representasi solusi dari permasalahan yang ingin diselesaikan dalam IGA. Kromosom terdiri atas sejumlah gen yang merupakan variabel-variabel solusi yang dapat digunakan untuk menyelesaikan permasalahan. Perepresentasian sebuah solusi menjadi kromosom merupakan hal yang vital karena hal tersebut akan mempengaruhi kualitas solusi yang dihasilkan oleh algoritma (Mahmudy, 2014).

Berikut contoh representasi kromosom :

Individu	**Kromosom**									
	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
1	1	10	13	3	10	15	4	8	11	1

2	4	10	13	4	7	13	4	9	14	4
3	3	10	14	5	9	13	2	9	14	3
4	2	9	11	5	7	13	3	7	15	2
5	2	10	12	4	9	13	3	6	11	2

Gambar 2.4 Contoh representasi kromosom

Gambar 2.4 menunjukkan 5 buah kromosom, dengan setiap individu memiliki kromosom dengan panjang 9 gen. Setiap gen memiliki nilai tertentu yang merepresentasikan solusi atas permasalahan yang ada.

2.6.2 Inisialisasi

Proses inisialisasi merupakan proses membentuk individu-individu generasi awal secara acak sesuai dengan representasi kromosom (Mahmudy, 2013). Generasi awal ini akan digunakan sebagai induk untuk mencari solusi paling optimal dari permasalahan yang ada.

2.6.3 Ekstraksi Vaksin

Proses ekstraksi vaksin digunakan untuk menyimpan individu dengan nilai *fitness* tertinggi untuk dijadikan vaksin (Yan-cong et al., 2011). Individu yang terpilih akan digandakan dan disimpan dalam *memory library* hingga tiba saatnya untuk melakukan vaksinasi atau digantikan dengan individu yang memiliki nilai *fitness* yang lebih tinggi.

Metode lain dalam melakukan ekstraksi vaksin ialah dengan menyimpan hanya sebagian kromosom dari satu individu (Jin & Fan, 2013). Sederet gen dalam satu kromosom dari individu terbaik akan disimpan dalam *memory library*. Deretan gen tersebut dapat dipilih dari titik tertentu sepanjang yang telah ditentukan dari satu kromosom individu terbaik.

Pada penelitian ini digunakan metode ekstraksi sebagian kromosom dari satu individu.

2.6.4 Perhitungan *fitness*

Perhitungan nilai *fitness* merupakan proses untuk menghitung kebugaran (*fitness*) kromosom (Mahmudy, 2013). Nilai *fitness* dapat menunjukkan tingkat baik-buruknya suatu kromosom.

Untuk menghitung nilai *fitness* dapat digunakan persamaan (2.1) (Juan et al., 2009)

$$f = \alpha C_1 + \beta C_2 + \delta C_3 + \varepsilon C_4 \quad \textbf{(2.1)}$$

Dimana α, β, δ, dan ε merupakan variabel pembobot, dan $\alpha + \beta + \delta + \varepsilon = 1$. Serta C_n merupakan variabel biaya.

2.6.5 Crossover

Crossover merupakan operator genetik yang digunakan untuk melahirkan kromosom baru yang mewarisi sifat-sifat induknya sebagaimana proses reproduksi yang terjadi dalam kehidupan alam (Zukhri, 2014). Pemilihan individu-individu yang akan mengalami *crossover* didasarkan pada nilai probabilitas *crossover* yang sudah ditentukan sebelumnya. *Crossover* dilakukan dengan memilih sepasang kromosom, kemudian memilih satu titik gen yang sama lalu menukarkan antar segmen kromosom sehingga terbentuk individu-individu baru yang memiliki gen campuran antara kedua orangtuanya.

Parent 1	1	10	13	3	10	15	4	8	11
Parent 2	4	10	13	4	7	13	4	9	14

offspring 1	1	10	13	3	10	15	4	9	14
offspring 2	4	10	13	4	7	13	4	8	11

Gambar 2.5 Contoh proses *crossover*

Gambar 2.5 memperlihatkan proses *crossover* yang terjadi pada kromosom individu Parent 1 dan Parent 2 yang terjadi pada titik *crossover* pada gen ke 7, sehingga gen ke 7 dan seterusnya akan disilangkan dengan pasangannya untuk membentuk individu baru.

2.6.6 Mutasi

Mutasi merupakan operator genetik yang bertujuan untuk mengubah gen-gen tertentu dari sebuah kromosom (Zukhri, 2014). Proses mutasi digunakan untuk menjaga keragaman dari populasi gen yang ada dalam satu generasi. Pemilihan individu yang akan mengalami mutasi didasarkan pada nilai probabilitas mutasi yang sudah ditentukan sebelumnya. Mutasi dapat dilakukan dengan memilih satu atau lebih titik mutasi, kemudian mengubah gen pada titik tersebut secara random sesuai dengan kolam gen yang ada.

Parent	1	10	13	3	10	15	4	8	11
offspring	3	10	13	3	10	15	4	8	11

Gambar 2.6 Contoh proses mutasi

Gambar 2.6 memperlihatkan proses mutasi yang terjadi pada titik gen ke-1. Gen pada titik titik tersebut akan diubah secara random sesuai dengan batasan yang ada.

2.6.7 Vaksinasi

Vaksinasi merupakan proses untuk memodifikasi gen dari individu tertentu untuk meningkatkan kemungkinan mendapatkan nilai *fitness* yang lebih tinggi (Lei & Licheng, 2000).

Pada model vaksin yang menyimpan keseluruhan gen pada individu sebagai vaksin, proses vaksinasi dilakukan dengan mengubah gen demi gen pada individu terpilih sesuai dengan vaksin yang terpilih sampai pada taraf tertentu sehingga didapatkan nilai *fitness* yang lebih tinggi (Yan-cong et al., 2011).

Parent	1	10	13	3	10	15	4	8	11
Vaksin	4	10	13	4	7	13	4	9	14
offspring	4	10	13	4	7	13	4	8	11

Gambar 2.7 Contoh proses vaksinasi vaksin lengkap

Gambar 2.7 memperlihatkan proses vaksinasi kromosom *parent* yang terjadi pada gen ke 1 hingga 6. Gen-gen pada titik vaksinasi tersebut akan diubah sesuai dengan gen pada vaksin. Vaksinasi dilakukan dengan mengubah gen satu demi satu hingga didapatkan nilai *fitness* yang lebih baik dari sebelumnya.

Pada vaksin yang hanya menyimpan sebagian gen dari kromosom, maka gen pada kromosom individu terpilih yang sesuai dengan lokasi gen pada vaksin akan diubah sesuai dengan gen yang ada pada vaksin. Jika ternyata hasil setelah vaksinasi didapatkan nilai *fitness* yang lebih buruk, maka gen-gen pada individu tersebut tidak akan berubah (Jin & Fan, 2013).

Parent	1	10	13	3	10	15	4	8	11
Vaksin					7	13	2		
offspring	4	10	13	3	7	13	2	8	11

Gambar 2.8 Contoh proses vaksinasi vaksin sebagian

Gambar 2.8 menunjukkan proses vaksinasi dengan vaksin yang hanya memiliki sebagian gen. Gen yang sesuai pada kromosom *parent* akan langsung diubah sesuai dengan gen yang ada pada vaksin.

Pada penelitian ini digunakan proses ekstraksi dan vaksinasi untuk vaksin yang hanya menyimpan sebagian gen dari kromosom individu.

2.6.8 Seleksi Imun

Proses seleksi mengombinasikan antara kemiripan antar antibodi dengan nilai *fitness* (Juan et al., 2009). Dengan kata lain seleksi imun memperhitungkan nilai *affinity* atau kedekatan baik antara antigen (permasalahan) dengan antibodi (solusi), maupun antar sesama antibodi. Perhitungan kedekatan antara antigen dan atibodi dapat dicapai dengan menggunakan perhitungan nilai *fitness*,

sedangkan kedekatan antar sesama antibodi dapat dihitung dengan menghitung jarak antibodi satu dengan atibodi lainnya dengan metode *Euclidean distance*.

Langkah-langkah dari seleksi imun dapat dijabarkan sebagai berikut,

1. *Affinity*. Nilai *affinity* digunakan untuk mengevaluasi kadar kedekatan antar antibodi. Nilai *affinity* dapat dihitung dengan persamaan

$$A_{ij} = \frac{1}{1 + \sqrt{\sum_{v=1}^{m}(i_v - j_v)^2}} \quad \textbf{(2.2)}$$

   Dimana A_{ij} merupakan nilai kedekatan antar antibodi *i* dan *j*, dan *m* merupakan banyak antibodi dalam satu populasi

2. Dari nilai *affinity* dapat dicari nilai kesamaan (similarity), yang dapat dihitung dengan persamaan,

$$C_i = \frac{banyak\ antibodi\ yang\ nilai\ kedekatannya\ terhadap\ antibodi\ i\ lebih\ besar\ daripada\ \phi}{N} \quad \textbf{(2.3)}$$

   Dimana N merupakan banyak antibodi dalam satu populasi, dan ϕ ialah konstanta *similarity* . Semakin besar nilai C, semakin mirip antara antibodi satu dan lainnya dalam satu populasi.

3. Menghitung probabilitas seleksi dengan persamaan

$$P = \mu P_f + (1 - \mu)P_d \quad \textbf{(2.4)}$$

   Dimana μ adalah konstanta *affinity*, Pf merupakan probabilitas berdasarkan nilai *fitness* yang dapat dicari dengan persamaan

$$P_f = \frac{f_i}{\sum_{i=1}^{N} f_i} \quad \textbf{(2.5)}$$

   Sedangkan Pd merupakan probabilitas berdasarkan nilai kesamaan yang dapat dicai dengan persamaan

$$P_d = \frac{C_i}{\sum_{i=1}^{N} C_i} \quad \textbf{(2.6)}$$

   Dimana N merupakan jumlah populasi, fi adalah nilai *fitness* antibodi, dan Ci adalah nilai *similarity* antibodi.

4. Menghitung banyak antibodi yang akan bertahan ke generasi selanjutnya.

$$N_r = N - N_d = N(1 - \sigma PD) \quad \textbf{(2.7)}$$

   Dimana Nr ialah jumlah antibodi yang akan bertahan ke generasi selanjutnya, N merupakan jumlah populasi saat ini, Nd adalah jumlah antibodi yang akan mati, σ adalah konstanta *death factor*, dan nilai PD dapat dicari dengan persamaan

$$PD = \frac{f_{avg}}{f_{max}} \quad \textbf{(2.8)}$$

Dimana favg merupakan rata rata nilai *fitness* dalam populasi, dan fmax merupakan nilai *fitness* tertinggi dalam populasi

2.6.9 Operasi *Update*

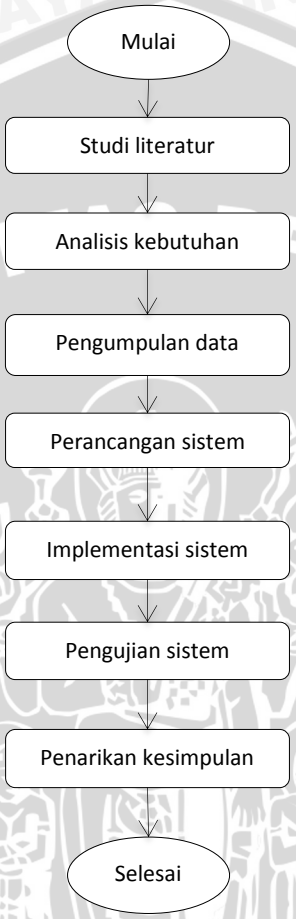
Operasi update merupakan proses untuk memperbarui generasi berdasarkan hasil probabilitas seleksi pada operasi imun. Pada proses kali ini terjadi proses seleksi menggunakan metode *roulette wheel selection* dengan nilai probabilitas sesuai dengan yang telah diperhitungkan pada operasi imun.

2.6.10 Kondisi Berhenti

Untuk menentukan kapan sebaiknya algoritma berhenti dapat dilakukan dengan melihat nilai *fitness* yang ada atau membatasi seberapa banyak jumlah iterasinya. Algoritma dapat dibatasi dengan nilai *fitness*, sehingga jika nilai *fitness* mencapai titik tertentu, maka algoritma dapat diberhentikan. Meskipun begitu metode tersebut tidak selamanya bisa tercapai, oleh karena itu diterapkannya metode kedua yaitu membatasi banyak iterasi, sehingga ketika sudah melebihi iterasi tertentu maka algoritma akan dipaksa berhenti dan menampilkan solusi terbaiknya saat itu.

BAB 3 METODOLOGI PENELITIAN

Metodologi penelitian menjelaskan metode yang akan digunakan dalam membuat sistem Optimasi Gizi pada Anak Panti Asuhan dengan *Immune Genetic Algorithm* pada Panti Asuhan di Kota Malang. Tahapan metodologi dapat dilihat pada Gambar 3.1,



Gambar 3.1 Tahapan metodologi penelitian

Berdasarkan Gambar 3.1, tahap tahap penelitan ini dapat dijabarkan sebagai berikut,

3.1 Studi Literatur

Pada tahap ini dilakukan pengumpulan referensi yang memiliki hubungan dengan optimasi gizi pada anak panti asuhan dengan *Immune Genetic Algorithm*, diantaranya :

1. Gizi
2. Panti Asuhan
3. Algoritma Genetika
4. *Artificial Immune System*
5. *Immune Genetic Algorithm*

Referensi tersebut dapat berasal dari jurnal nasional dan internasional, skripsi terdahulu, buku, serta artikel yang terdapat di internet. Selain itu juga dilakukan wawancara dengan pakar yang sesuai dengan bidang penelitian ini.

3.2 Analisis Kebutuhan

Analisis kebutuhan dilakukan dengan menentukan kebutuhan apa saja yang diperlukan untuk membangun sistem optimasi gizi pada anak panti asuhan dengan *Immune Genetic Algorithm*.

3.2.1 Kebutuhan fungsional

Analisis kebutuhan fungsional dilakukan untuk memberi gambaran proses apa yang dapat dilakukan oleh sistem.

1. Sistem dapat menerima input berupa nilai jumlah populasi, probabilitas *crossover*, probabilitas mutasi, konstanta *affinity*, konstanta *similarity*, dan jumlah maksimum iterasi.
2. Sistem dapat menampilkan input yang dilakukan oleh user
3. Sistem dapat menampilkan data menu makanan beserta rincian gizi dan harganya kepada user
4. Sistem dapat menampilkan perhitungan IGA sesuai dengan tahap tahapnya.
5. Sistem dapat menampilkan hasil solusi yang paling optimal

3.2.2 Kebutuhan non fungsional

Analisis kebutuhan non fungsional dilakukan untuk mengetahui spesifikasi kebutuhan untuk sistem. Spesifikasi kebutuhan melibatkan analisis perangkat keras, analisis perangkat lunak, serta analisis data yang diperlukan

1. Kebutuhan perangkat keras
    - Monitor LCD 1766 * 768
    - Keyboard dan Mouse/Touchpad
2. Kebutuhan perangkat lunak
    - Sistem operasi Windows 7
    - Java JRE 8
3. Kebutuhan data
    - Data anak panti asuhan
    - Data menu makanan panti asuhan
    - Data harga menu makanan panti asuhan
    - Data kandungan gizi menu makanan

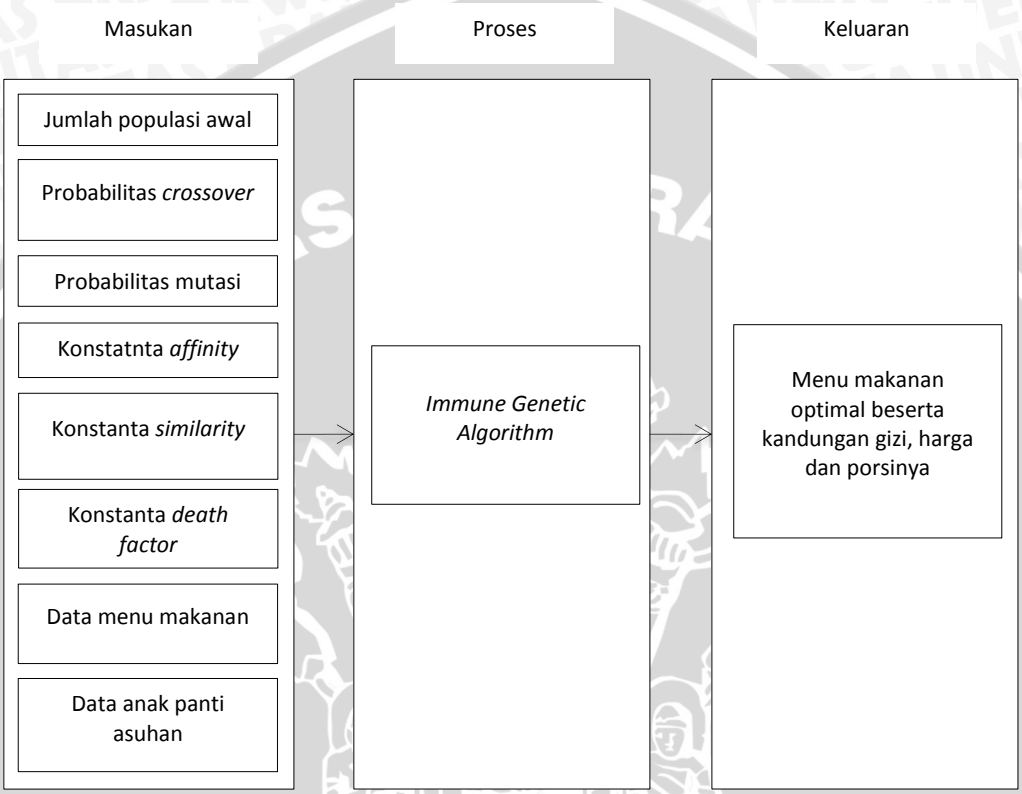
3.3 Pengumpulan Data

Pengumpulan data dilakukan dengan mendapatkan data terkait penelitian. Data anak panti asuhan, dan data menu makanan panti asuhan didapatkan dari panti asuhan, yakni dengan melakukan wawancara kepada pengurus panti asuhan. Data kandungan gizi menu makanan didapatkan dari mencocokkan data menu makanan dengan buku Daftar Komposisi Bahan Makanan dan Daftar

Bahan Makanan Pengganti. Data harga menu makanan didapatkan dengan survey lapangan.

3.4 Perancangan Sistem

Perancangan sistem dibuat untuk rancangan langkah kerja sistem secara menyeluruh. Model perancangan pada Gambar 3.2 menunjukkan cara kerja sistem secara terstruktur mulai dari input, proses, hingga output.



Gambar 3.2 Model perancangan sistem

Pada Gambar 3.2 terdapat tiga proses utama, yaitu:

1. Masukan. Masukan yang harus dimasukan oleh pengguna pada sistem ini ialah jumlah populasi awal, probabilitas *crossover*, probabilitas mutasi, konstanta *affinity*, konstanta *similarity*, konstanta *death factor*, data menu makanan, serta data anak panti asuhan.
2. Proses. Penggunaan *Immune Genetic Algorithm* untuk pencarian menu makanan paling optimal bagi anak panti asuhan dilihat dari kebutuhan gizi, serta harga makanan tersebut.
3. Keluaran. Keluaran dari sistem ini berupa menu makanan yang paling optimal beserta dengan kandungan gizi dan harga sesuai hasil optimasi *Immune Genetic Algorithm*.

3.5 Implementasi Sistem

Implementasi sistem merupakan fase pembangunan sistem yang mengacu pada perancangan sistem dengan menerapkan pengetahuan yang telah didapat dalam proses studi literatur. Proses-proses yang terdapat pada fase ini antara lain,

1. Implementasi *Immune Genetic Algorithm*. Algoritma ini akan diimplementasikan menggunakan bahasa pemrograman Java dengan pendekatan *Object Oriented Programming*.
2. Implementasi antarmuka. Antarmuka sistem akan menggunakan antarmuka SWING yang merupakan antarmuka yang telah disediakan oleh Java, khususnya pada IDE Netbeans. Antarmuka sistem digunakan agar pengguna lebih mudah dalam melakukan interaksi dengan sistem, baik melakukan masukan maupun melihat proses dan hasil algoritma.

3.6 Pengujian Algoritma

Pada tahap ini dilakukan pengujian sistem optimasi gizi anak panti asuhan dengan *immune genetic algorithm* yang telah diimplementasikan. Pengujian dilakukan untuk mengetahui pengaruh parameter-parameter dari *immune genetic algorithm* terhadap hasil keluaran. Pengujian ini meliputi,

1. Pengujian jumlah populasi awal
2. Pengujian probabilitas *crossover*
3. Pengujian probabilitas mutasi
4. Pengujian konstanta *affinity*
5. Pengujian konstanta *similarity*
6. Pengujian konstanta *death factor*

Berikut masing-masing skenario untuk pengujian algoritma

3.6.1 Pengujian jumlah populasi awal

Pengujian jumlah populasi awal ditujukan untuk mengetahui populasi awal yang paling optimal untuk menyelesaikan permasalahan optimasi gizi anak panti asuhan dengan *immune genetic algorithm*. Jumlah populasi awal yang diuji ialah mulai dari 10 hingga 100 dengan pengujian pada kelipatan 10. Parameter yang digunakan pada pengujian jumlah populasi awal ialah sebagai berikut,

1. Banyak iterasi = 1000
1. Jumlah populasi awal = 10 – 100
2. probabilitas *crossover* = 0,6
3. probabilitas mutasi = 0,1
4. konstanta *affinity* = 0,3
5. konstanta *similarity* = 0,9
6. konstanta *death factor* = 0,5

Pengujian dilakukan 5 kali untuk masing-masing jumlah populasi awal yang diujikan. Hasil pengujian dicatat pada Tabel 3.1

Tabel 3.1 Pengujian jumlah populasi awal

Populasi		Uji coba ke-					Max iterasi	Rata iterasi	Min iterasi	Rata *fitness*
		1	2	3	4	5				
	Konvergen									
10	*Fitness*									
	k									
20	f									
	k									
...	f									
	k									
100	f									

Dengan k = konvergen pada iterasi ke- , dan f = nilai *fitness* saat konvergen

3.6.2 Pengujian probabilitas *crossover*

Pengujian probabilitas *crossover* ditujukan untuk mengetahui nilai probabilitas *crossover* yang paling optimal dalam penyelesaian optimasi gizi anak panti asuhan dengan *immune genetic algorithm*. Nilai probabilitas *crossover* yang diuji coba mulai dari 0 hingga 1 dengan kelipatan 0,1 tiap pengujian. Parameter yang digunakan dalam pengujian banyak iterasi ini ialah sebagai berikut,

1. Banyak iterasi = 1000
2. Jumlah populasi awal = nilai terbaik pada pengujian jumlah populasi awal
3. probabilitas *crossover* = 0 – 1
4. probabilitas mutasi = 0,1
5. konstanta *affinity* = 0,3
6. konstanta *similarity* = 0,9
7. konstanta *death factor* = 0,5

Pengujian dilakukan 5 kali untuk masing-masing probabilitas *crossover* yang diujikan. Hasil pengujian dicatat pada Tabel 3.2

Tabel 3.2 Pengujian probabilitas *crossover*

Probabilitas *crossover*		Uji coba ke-					Max iterasi	Rata iterasi	Min iterasi	Rata *fitness*
		1	2	3	4	5				
	k									
0	f									
	k									
0,1	f									
	k									
...	f									
	k									
1	f									

Dengan k = konvergen pada iterasi ke- , dan f = nilai fitness saat konvergen

3.6.3 Pengujian probabilitas mutasi

Pengujian probabilitas mutasi ditujukan untuk nilai probabilitas mutasi yang paling optimal untuk menyelesaikan permasalahan optimasi gizi anak panti asuhan dengan *immune genetic algorithm*. Nilai probabilitas mutasi yang diuji ialah mulai dari 0 hingga 1 dengan kelipatan 0,1 tiap pengujian. Parameter yang digunakan pada pengujian nilai probabilitas mutasi ialah sebagai berikut,

1. Banyak iterasi = 1000
2. Jumlah populasi awal = nilai terbaik pada pengujian jumlah populasi awal
3. probabilitas *crossover* = nilai terbaik pada pengujian probabilitas *crossover*
4. probabilitas mutasi = 0 – 1
5. konstanta *affinity* = 0,3
6. konstanta *similarity* = 0,9
7. konstanta *death factor* = 0,5

Pengujian dilakukan 5 kali untuk masing-masing probabilitas mutasi yang diujikan. Hasil pengujian dicatat pada Tabel 3.3

Tabel 3.3 Pengujian probabilitas mutasi

Probabilitas mutasi		Uji coba ke-					Max iterasi	Rata iterasi	Min iterasi	Rata *fitness*
		1	2	3	4	5				
0	k									
	f									
0,1	k									
	f									
...	k									
	f									
1	k									
	f									

Dengan k = konvergen pada iterasi ke- , dan f = nilai fitness saat konvergen

3.6.4 Pengujian konstanta *affinity*

Pengujian konstanta *affinity* ditujukan untuk mengetahui pengaruh nilai konstanta *affinity* dalam penyelesaian optimasi gizi anak panti asuhan dengan *immune genetic algorithm*. Nilai konstanta *affinity* yang diuji coba mulai dari 0 hingga 1 dengan kelipatan 0,1 tiap pengujian. Parameter yang digunakan dalam pengujian konstanta *affinity* ini ialah sebagai berikut,

1. Banyak iterasi = 1000
2. Jumlah populasi awal = nilai terbaik pada pengujian jumlah populasi awal
3. probabilitas *crossover* = nilai terbaik pada pengujian probabilitas *crossover*
4. probabilitas mutasi = nilai terbaik pada pengujian probabilitas mutasi
5. konstanta *affinity* = 0 – 1
6. konstanta *similarity* = 0,9
7. konstanta *death factor* = 0,5

Pengujian dilakukan 5 kali untuk masing-masing konstanta *affinity* yang diujikan. Hasil pengujian dicatat pada Tabel 3.4

Tabel 3.4 Pengujian konstanta *affinity*

Konstanta *affinity*		Uji coba ke- 1	2	3	4	5	Max iterasi	Rata iterasi	Min iterasi	Rata *fitness*
0	k									
	f									
0,1	k									
	f									
...	k									
	f									
1	k									
	f									

Dengan k = konvergen pada iterasi ke- , dan f = nilai fitness saat konvergen

3.6.5 Pengujian konstanta *similarity*

Pengujian konstanta *similarity* ditujukan untuk mengetahui pengaruh nilai konstanta *similarity* dalam penyelesaian optimasi gizi anak panti asuhan dengan *immune genetic algorithm*. Nilai konstanta *similarity* yang diuji coba mulai dari 0,1 hingga 1 dengan kelipatan 0,1 tiap pengujian. Parameter yang digunakan dalam pengujian konstanta *similarity* ini ialah sebagai berikut,

1. Banyak iterasi = 1000
2. Jumlah populasi awal = nilai terbaik pada pengujian jumlah populasi awal
3. probabilitas *crossover* = nilai terbaik pada pengujian probabilitas *crossover*
4. probabilitas mutasi = nilai terbaik pada pengujian probabilitas mutasi
5. konstanta *affinity* = nilai terbaik pada pengujian konstanta *affinity*
6. konstanta *similarity* = 0,1 – 1
7. konstanta *death factor* = 0,5

Pengujian dilakukan 5 kali untuk masing-masing konstanta *similarity* yang diujikan. Hasil pengujian dicatat pada Tabel 3.5

Tabel 3.5 Pengujian konstanta *similarity*

Konstanta *similarity*		Uji coba ke- 1	2	3	4	5	Max iterasi	Rata iterasi	Min iterasi	Rata *fitness*
0,1	k									
	f									
0,2	k									
	f									
...	k									
	f									
1	k									
	f									

Dengan k = konvergen pada iterasi ke- , dan f = nilai fitness saat konvergen

3.6.6 Pengujian konstanta *death factor*

Pengujian konstanta *death factor* ditujukan untuk mengetahui pengaruh nilai konstanta *death factor* dalam penyelesaian optimasi gizi anak panti asuhan dengan *immune genetic algorithm*. Nilai konstanta *death factor* yang diuji coba mulai dari 0 hingga 1 dengan kelipatan 0,1 tiap pengujian. Parameter yang digunakan dalam pengujian *death factor* ini ialah sebagai berikut,

1. Banyak iterasi = 1000
2. Jumlah populasi awal = nilai terbaik pada pengujian jumlah populasi awal
3. probabilitas *crossover* = nilai terbaik pada pengujian probabilitas *crossover*
4. probabilitas mutasi = nilai terbaik pada pengujian probabilitas mutasi
5. konstanta *affinity* = nilai terbaik pada pengujian konstanta *affinity*
6. konstanta *similarity* = nilai terbaik pada pengujian konstanta *similarity*
7. konstanta *death factor* = 0,1 – 1

Pengujian dilakukan 5 kali untuk masing-masing konstanta *death factor* yang diujikan. Hasil pengujian dicatat pada Tabel 3.6

Tabel 3.6 Pengujian konstanta *death factor*

Konstanta *death factor*		Uji coba ke-					Max iterasi	Rata iterasi	Min iterasi	Rata *fitness*
		1	2	3	4	5				
0,1	k									
	f									
0,2	k									
	f									
...	k									
	f									
1	k									
	f									

Dengan k = konvergen pada iterasi ke- , dan f = nilai fitness saat konvergen

3.7 Penarikan Kesimpulan

Penarikan kesimpulan dilakukan setelah semua tahap perancangan, implementasi, dan pengujian telah selesai dilakukan. Kesimpulan diambil berdasarkan hasil dari pengujian sistem. Kesimpulan dibuat dengan tujuan untuk menjawab rumusan masalah yang telah disampaikan pada Bab I. Tahap terakhir dari penulisan ialah saran yang dimaksudkan untuk memperbaiki kekurangan yang ada serta memberikan pertimbangan untuk pengembangan penelitian selanjutnya.

BAB 4 PERANCANGAN

Bab ini menjelaskan formulasi permasalahan, siklus *Immune Genetic Algorithm*, siklus penyelesaian optimasi gizi pada anak panti asuhan dengan *Immune Genetic Algorithm*, perhitungan manual, serta perancangan antar muka.

4.1 Formulasi permasalahan

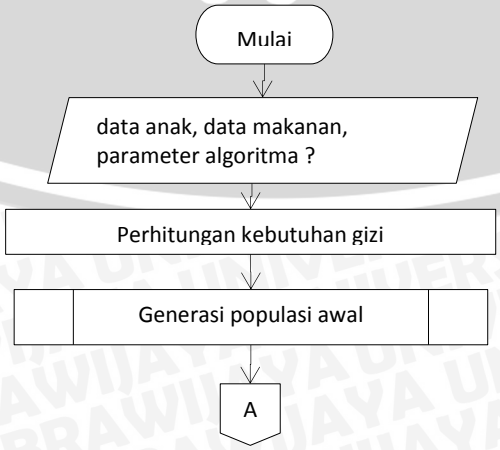
Permasalahan yang ingin diselesaikan dalam penelitian ini ialah optimasi gizi untuk anak panti asuhan. Optimasi gizi yang dimaksud ialah mengoptimasi menu makanan pada panti asuhan agar dapat memenuhi kebutuhan gizi anak panti asuhan sehingga dapat mencegah terjadinya permasalahan gizi pada anak-anak tersebut baik gizi kurang maupun gizi berlebih.

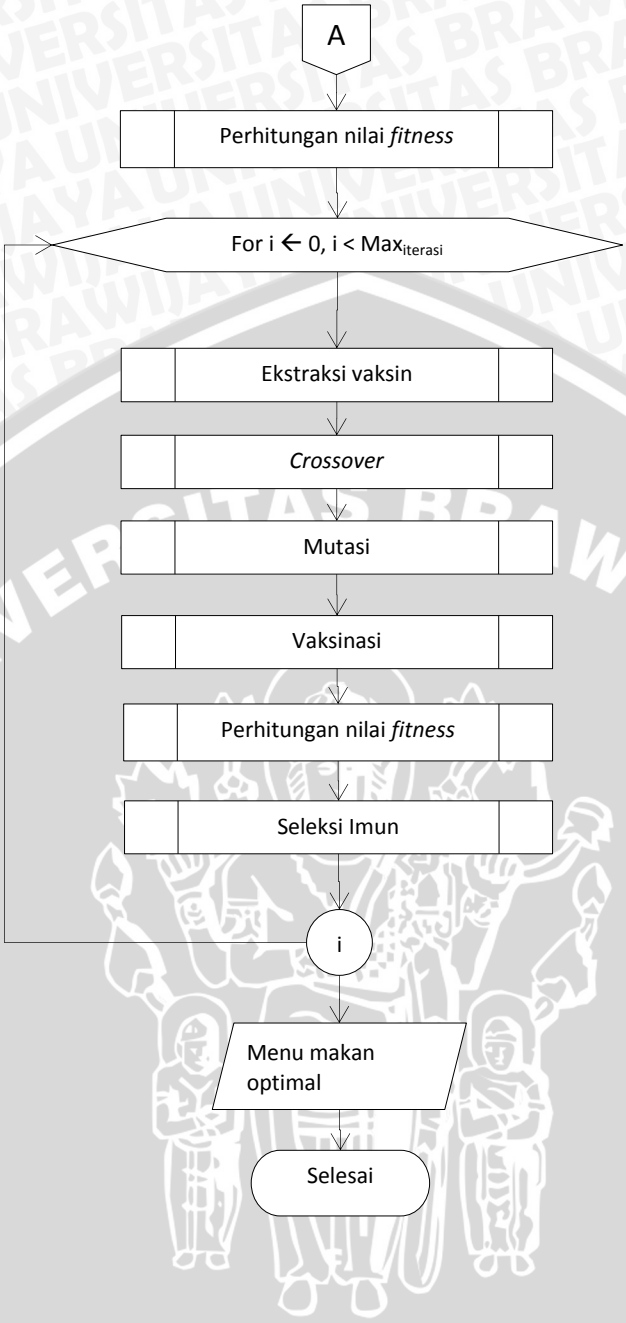
Hasil solusi dari optimasi tersebut ialah daftar menu makanan untuk satu minggu (7 hari) yang paling optimum dilihat dari jumlah gizi yang dikandung oleh setiap makanan dibandingkan dengan total kebutuhan gizi anak-anak panti asuhan dengan mempertimbangkan total biaya yang diperlukan serta variasi menu makanannya.

Masukan yang dibutuhkan oleh sistem antara lain data anak panti asuhan dalam format .xls, data makanan dalam format .xls, dan parameter-parameter perhitungan meliputi ukuran populasi awal, probabilitas *crossover*, probabilitas mutasi, konstanta *affinity*, konstanta *similarity*, dan konstanta *death factor*. Selanjutnya masuk proses inisialisasi awal kemudian proses iterasi *Immune Genetic Algorithm*. Proses iterasi dilakukan hingga menemukan solusi yang paling optimal atau hingga mencapai batas iterasi tertentu.

4.2 Siklus *Immune Genetic Algorithm*

Siklus *Immune Genetic Algorithm* ialah urutan penyelesaian permasalahan dengan menggunakan *Immune Genetic Algorithm*. Dalam penyelesaian permasalahan akan dilakukan beberapa proses dalam siklus algoritma. Secara garis besar, siklus *Immune Genetic Algorithm* yang akan digunakan pada penelitian ini dapat dilihat pada Gambar 4.1.





Gambar 4.1 Siklus *Immune Genetic Algorithm*

Langkah langkah optimasi menggunakan *Immune Genetic Algorithm* berdasarkan Gambar 4.1 ialah sebagai berikut,

1. Sistem menerima masukan berupa data anak panti asuhan yang meliputi umur, jenis kelamin, dan berat badan, data menu makananSiklus *Immune Genetic Algorithm* beserta dengan kandungan gizi dan harganya, serta parameter algoritma yang meliputi ukuran populasi awal, probabilitas *crossover* (*cr*), probabilitas mutasi (*mr*), batas iterasi, batas harga, ukuran *immune memory*, konstanta *similarity* (ϕ), konstanta *affinity* (μ), serta *death factor*(σ).

2. Perhitungan kebutuhan gizi anak anak panti asuhan
3. Perhitungan nilai *fitness*
4. Untuk iterasi 1 hingga batas iterasi maksimal atau hingga tercapai solusi optimum, lakukan langkah 5 hingga 10.
5. Melakukan ekstraksi vaksin
6. Melakukan proses *crossover*
7. Melakukan proses mutasi
8. Melakukan proses vaksinasi
9. Melakukan proses perhitungan nilai *fitness*
10. Melakukan proses seleksi imun
11. Hasil keluaran ialah menu makanan untuk banyak hari sesuai masukan pengguna, dengan kebutuhan gizi dan harga yang paling optimum.

Pada bab ini akan dijelaskan siklus dan penghitungan untuk *immune genetic algorithm* secara garis besar, untuk detail perhitungan manual dapat dilihat pada Lampiran A Perhitungan Manual.

4.2.1 Perhitungan kebutuhan gizi

Kebutuhan gizi tiap anak panti asuhan dihitung dengan berdasarkan tabel Angka Kecukupan Gizi, kemudian dijumlah untuk mendapatkan kebutuhan total dari keseluruhan anak panti asuhan. Kebutuhan gizi tersebut meliputi kebutuhan energi, protein, lemak, dan karbohidrat.

Dari kebutuhan gizi total tersebut, akan dibagi lagi menjadi kebutuhan tiap makan, baik makan pagi, siang, dan malam, yang dapat dihitung dengan persamaan berikut masing-masing untuk energi (4.1), protein (4.2), lemak (4.3), dan karbohidrat (4.4).

$$Energi\ tiap\ makan = \begin{cases} Kebutuhan\ energi\ * 25\% & , pagi\ atau\ malam \\ Kebutuhan\ energi\ * 30\% & , siang \end{cases} \quad \textbf{(4.1)}$$

$$Protein\ tiap\ makan = \begin{cases} Kebutuhan\ protein\ * 25\% & , pagi\ atau\ malam \\ Kebutuhan\ protein\ * 30\% & , siang \end{cases} \quad \textbf{(4.2)}$$

$$Lemak\ tiap\ makan = \begin{cases} Kebutuhan\ lemak\ * 25\% & , pagi\ atau\ malam \\ Kebutuhan\ lemak\ * 30\% & , siang \end{cases} \quad \textbf{(4.3)}$$

$$Karbohidrat\ tiap\ makan = \begin{cases} Kebutuhan\ karbohidrat\ * 25\% & , pagi\ atau\ malam \\ Kebutuhan\ karbohidrat\ * 30\% & , siang \end{cases} \quad \textbf{(4.4)}$$

Contoh perhitungan manual dari perhitungan kebutuhan gizi dapat dilihat pada studi kasus berikut :
Jika terdapat 3 anak laki laki pada rentang usia 10 – 12 tahun, dan 7 anak perempuan pada rentang umur 13 – 15 tahun maka,
Kebutuhan energi total = (3*2100) + (7*2125)

= 6300 + 14875
= 21175

Dimana nilai 2100 dan 2125 didapat dari nilai kebutuhan energi pada tabel AKG sesuai dengan rentang umur dan jenis kelamin yang sama, nilai 3 didapat dari jumlah anak laki laki pada rentang umur 10-12 tahun, nilai 7 didapat dari jumlah anak pada rentang umur 13-15 tahun.

Perhitungan kebutuhan total untuk protein, lemak, dan karbohidrat didapat dengan cara yang hampir sama dengan perhitungan energi total dengan menyesuaikan kebutuhannya.

Dari kebutuhan total tersebut akan dihitung lagi untuk kebutuhan tiap makan, baik itu pagi, siang, atau malam. Perhitungan kebutuhan energi tiap makan menggunakan persamaan (4.1).

Kebutuhan energi pagi = 21175 * 25%
= 21175 * 0.25
= 5293.75

Kebutuhan energi siang = 21175 * 30%
= 21175 * 0.3
= 6352.5

Kebutuhan energi malam = 21175 * 25%
= 21175 * 0.25
= 5293.75

Perhitungan kebutuhan tiap makan untuk protein, lemak, dan karbohidrat dihitung dengan cara yang hampir sama dengan menggunakan persamaan (4.2), (4.3), dan (4.4).

4.2.2 Generasi populasi awal

Generasi populasi awal dilakukan dengan membuat antibodi antibodi baru sebanyak jumlah populasi awal yang diinginkan. Setiap antibodi direpresentasikan dengan kromosom yang menyatakan menu makanan. Kromosom tersebut memiliki panjang 84 gen, yang dibagi menjadi 21 fragmen. Tiap fragmen memiliki panjang 4 gen yang merepresentasikan menu makanan dalam satu kali makan baik pagi, siang, maupun malam yang meliputi makanan pokok, sayur, lauk, dan minuman. Setiap tiga fragmen yang berurutan merupakan menu makanan dalam satu hari dimana fragmen pertama untuk makan pagi, fragmen kedua untuk makan siang, dan fragmen ketiga untuk makan malam. Sehingga ke-84 gen tersebut akan merepresentasikan menu makanan selama satu minggu (7 hari). Skema representasi kromosom dapat dilihat pada Gambar 4.2 berikut.

d_i	d_{i+1}	d_{i+2}	d_{i+3}	d_{i+4}	d_{i+5}	d_{i+6}	d_{i+7}	$d_{i+...}$	d_{i+83}

Gambar 4.2 Representasi kromosom

Keterangan :
d = data menu makanan
i = data ke-*i*

dengan data ke i mod 4 = 0 adalah data makanan pokok, data ke i mod 4 = 1 adalah data sayur, i mod 4 = 2 adalah data lauk, i mod 4 = 3 adalah data minuman, serta data ke i dengan i mod 12 < 4 adalah kelompok data makan pagi, data ke i dengan 4 <= i mod 12 < 8 adalah kelompok data makan siang, dan data ke i dengan i mod 12 >= 8 untuk kelompok data makan malam.

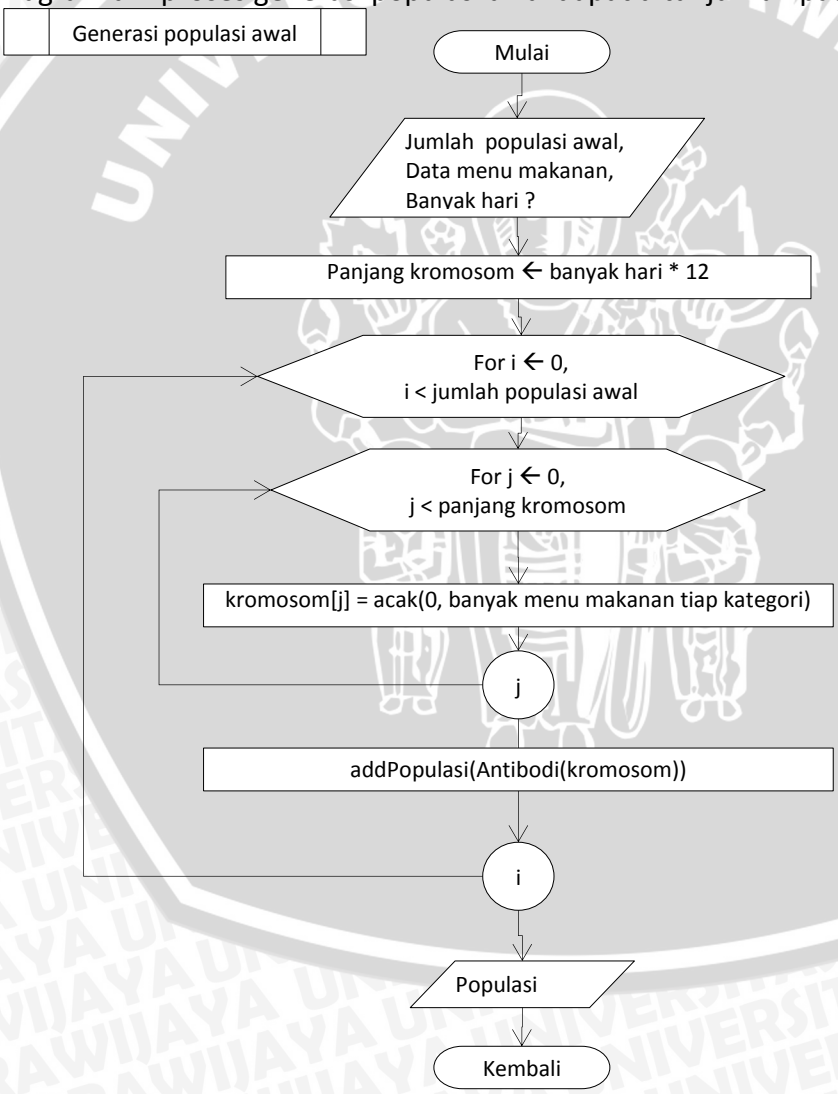
Kromosom tersebut menggunakan pengkodean real dengan menggunakan bilangan integer untuk menunjukkan indeks menu makanan.

Contoh hasil generasi kromosom dapat dilihat pada Gambar 4.3

3	*8*	*8*	*1*	3	4	1	*1*	...	*2*

Gambar 4.3 Contoh kromosom

Diagram alir proses generasi populasi awal dapat ditunjukkan pada Gambar 4.4



Gambar 4.4 Diagram alir generasi populasi awal

4.2.3 Perhitungan nilai *fitness*

Tiap antibodi yang dihasilkan akan dihitung nilai *fitness*-nya sebagai tolak ukur seberapa baik antibodi tersebut terhadap antigen atau permasalahan. Nilai *fitness* tiap antibodi didapatkan dari jumlah penalti gizi dengan nilai variasi dan harga makanan. Secara umum, persamaan nilai *fitness* ditunjukkan pada persamaan (4.5) berikut,

$$fitness = \frac{1}{penalti_{protein} + 1} + \frac{1}{penalti_{lemak} + 1} + \frac{1}{penalti_{karbohidrat} + 1} + \frac{1}{variasi_{kesamaan} + 1} + \frac{1000}{penalti_{biaya} + 1} \quad \textbf{(4.5)}$$

Nilai penalti didapatkan dari selisih antara kebutuhan gizi dengan gizi yang diberikan oleh menu pada kromosom. Persamaan (4.6) menunjukkan perhitungan penalti protein.

$$penalti_{protein} = \sum_{i=0}^{n} \left| \left[\sum_{j=i*4}^{j=(1*4)+4} (protein_j * jumlah\ anak) \right] - kebutuhan_{protein} \right| \quad \textbf{(4.6)}$$

Keterangan :

$protein_j$ = menunjukkan nilai kandungan protein pada menu makanan j

$faktorPengali_j$ = menunjukkan nilai faktor pengali pada gen j

$kebutuhan_{protein}$ = menunjukkan kebutuhan protein untuk sekali makan, baik pagi, siang, ataupun malam

n = menunjukkan banyak fragmen atau banyak waktu makan

Penalti lemak dan karbohidrat dapat dicari dengan cara yang sama dengan protein, menyesuaikan nilai gizi yang digunakan apakah itu protein, lemak, karbohidrat.

Untuk mencari nilai penalti harga, dapat dengan menjumlahkan harga dari tiap menu makanan dikalikan faktor pengali. Jumlah dari selisih dari batas harga dan biaya per hari merupakan nilai penalti harga. Sehingga perhitungan penalti harga dapat dituliskan pada persamaan (4.7).

$$Selisih_{biaya} = \begin{cases} biaya * jumlah_{anak} - batas_{biaya}, & biaya * jumlah_{anak} > batas_{biaya} \\ 0, biaya * jumlah_{anak} \leq batas_{biaya} \end{cases}$$

$$penalti_{biaya} = \sum_{i=0}^{n} Selisih_{biaya} \quad \textbf{(4.7)}$$

Keterangan:

$Batas_{biaya}$ = menunjukkan batas biaya pembelanjaan yang diharapkan dalam satu hari

$Selisih_{biaya}$ = menunjukkan selisih antara biaya menu dengan batas biaya pembelanjaan yang diharapkan dalam satu hari

n = menunjukkan banyak fragmen atau banyak waktu makan

Sedangkan variasi kesamaan menunjukkan banyaknya menu yang sama setiap kali makan selama 7 hari. Setiap menu pada setiap fragmen akan dibandingkan dengan menu pada fragmen lain sesuai dengan jenis menunya, jika dijumpai menu yang sama antar fragmen, nilai variasi kesamaan akan bertambah..

Contoh perhitungan manual dari proses perhitungan nilai *fitness* yaitu,

Didapatkan kromosom dari individu berikut,

3	8	8	1	3	4	1	1	3	10	7	4

Dari kromosom tersebut diketahui kandungan protein, lemak dan karbohidratnya, serta harganya untuk porsi sejumlah banyak anak yang dimasukkan.

kromosom	3	8	8	1	3	4	1	1	3	10	7	4
protein	357	81	76	0	357	276	333	0	357	105	142	0
lemak	353	73	87	0	353	126	221	0	353	357	118	0
karbohidrat	1192	455	68	0	1192	1530	164	0	1192	164	85	699
harga	95000	38000	11400	3800	95000	57000	38000	3800	95000	38000	19000	11400

Dari kandungan tiap makanan, akan dijumlahkan sehingga didapat kandungan gizi tiap kali makan,

Kandungan protein sekali makan (pagi) = 357 + 81 + 76 + 0

= 513

Dengan cara yang sama dihitung untuk waktu makan dan gizi yang lainnya, sehingga didapatkan nilai sebagai berikut,

	pagi	siang	malam
protein	513	966	603
lemak	513	700	828
karbohidrat	1716	2886	2140
harga	148200	193800	163400

Dari kandungan gizi tiap makan, bisa dicari nilai selisih kebutuhan gizi,

Selisih protein (pagi) = | 513 – 600 |

= 87

Dimana nilai 600 merupakan kebutuhan protein untuk satu kali makan pada waktu yang sama (pagi). Nilai selisih untuk kandungan gizi yang lain dapat dicari dengan cara yang sama. Jika ternyata selisih gizinya kurang dari batas toleransi 10% kebutuhan gizi, maka selisih gizi akan menjadi 0.

Sedangkan untuk harga,

Selisih harga (pagi) = | 148200– 50000 |

= 98200

Nilai 50000 didapat dari masukan pengguna sebagai preferensi berapa biaya yang diinginkan untuk satu kali makan. Dari perhitungan selisih, didapatkan hasil berikut,

protein	87	245	3
lemak	211	169	103
karbohidrat	1257	681	832
harga	98200	143800	113400

Nilai penalti adalah jumlah dari nilai selisih untuk tiap gizi maupun harga,

Penalti protein = 87 + 245 + 3

= 335

Penalti untuk nilai gizi dan harga dapat dicari dengan cara yang sama, sehingga didapatkan nilai penalti sebagai berikut,

protein	335
lemak	483
karbohidrat	2770
harga	355400

Untuk mencari nilai variasi kesamaan, dapat diilustrasikan seperti berikut,

Kromosom	3	8	8	1	3	4	1	1	3	10	7	4
3	0				1				1			
8		0				0				0		
8			0				0				0	
1				0				1				0
3	1				0				1			
4		0				0				0		
1			0				0				0	
1				1				0				0
3	1				1				0			
10		0				0				0		
7			0				0				0	
4				0				0				0
3	1				1				1			
4		0				1				0		
2			0				0				0	
2				0				0				0

Kromosom akan dibandingkan dengan dirinya sendiri. Nilai 0 menunjukkan jika antara baris dan kolom memiliki nilai yang berbeda, atau merupakan gen dengan posisi yang sama pada kromosom, sedangkan nilai 1 menunjukkan jika antara baris dan kolom memiliki nilai yang sama. Perbandingan dilakukan jika jenis gen sama, yaitu sesama makanan pokok, lauk, sayur, atau minuman. Nilai kesamaan dihitung dari jumlah banyaknya angka satu (1).

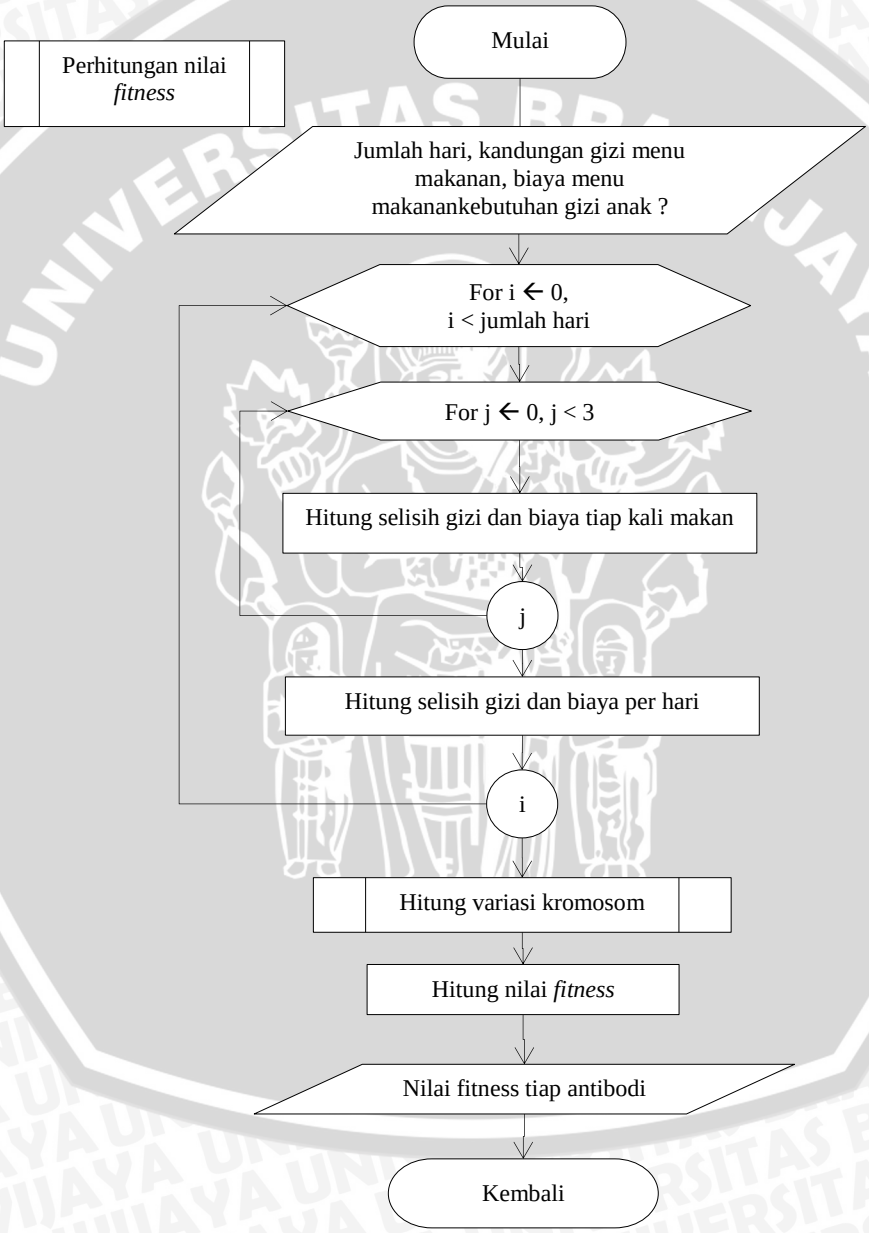
kesamaan	12

Setelah didapatkan nilai penalti dan nilai kesamaan, dapat dicari nilai *fitness* untuk kromosom tersebut.

fitness = (1/335) + (1/483) + (1/2770) + (1000/355400) + (1/12)

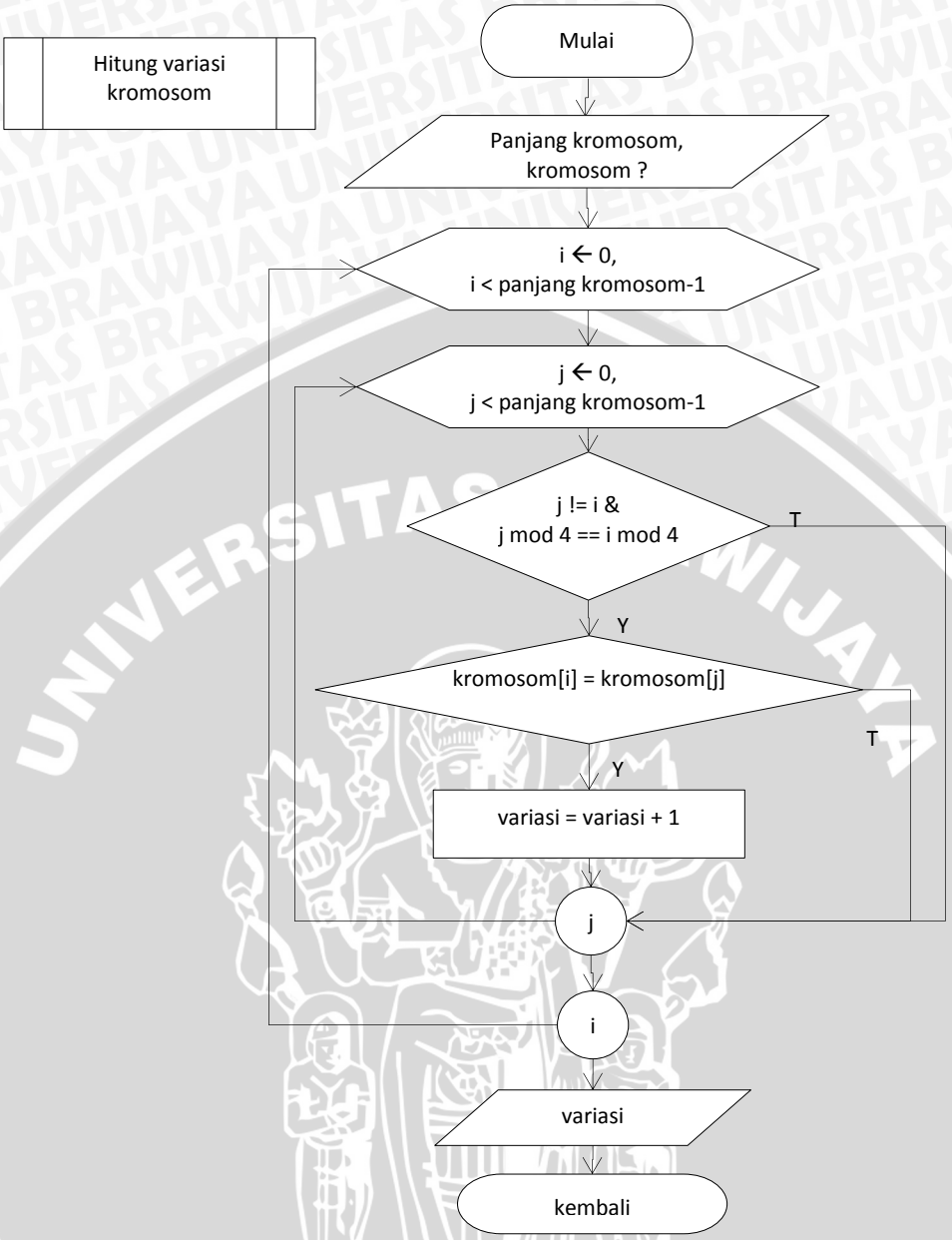
= 0.091

Diagram alir untuk proses perhitungan nilai *fitness* dapat dilihat pada Gambar 4.5.



Gambar 4.5 Diagram alir perhitungan nilai *fitness*

Sedangkan proses perhitungan variasi kromosom, dapat dilihat pada Gambar 4.6.



Gambar 4.6 Diagram alir perhitungan variasi kromosom

4.2.4 Ekstraksi Vaksin

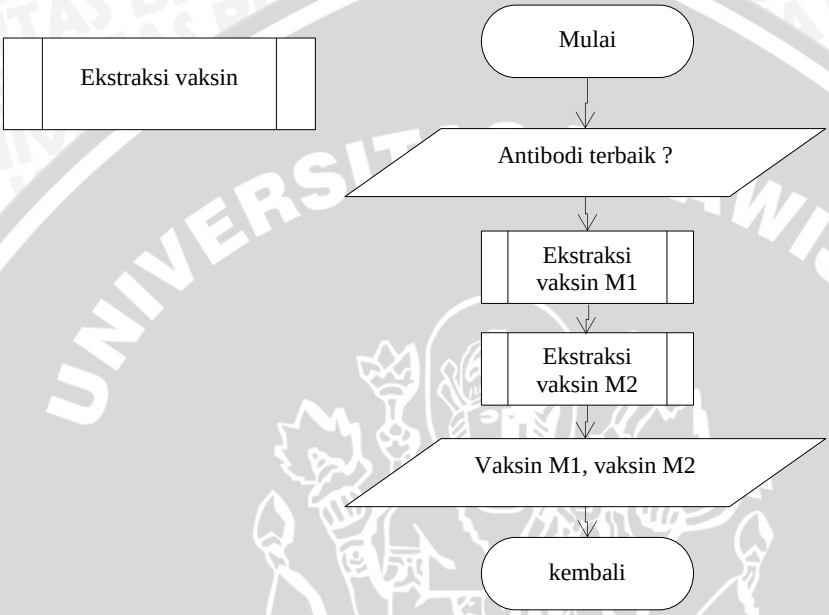
Metode yang digunakan untuk ekstraksi vaksin pada penelitian ini ialah dengan menyimpan sebagian kromosom dari individu. Proses ekstraksi vaksin dilakukan dalam dua tahap, yaitu ekstraksi vaksin M1 dan ekstraksi vaksin M2, yang keduanya masing-masing akan menjadi satu vaksin utuh, sehingga dalam satu kali ekstraksi vaksin akan dihasilkan dua buah vaksin. Tiap vaksin yang dihasilkan akan disimpan dalam *immune memory* untuk menggantikan vaksin yang memiliki *fitness* lebih rendah.

Pemilihan kandidat antibodi berdasarkan antibodi dengan nilai *fitness* tertinggi dalam populasi saat itu. Dari antibodi terpilih akan diproses untuk ekstraksi baik untuk ekstraksi M1 maupun M2. Proses ekstraksi vaksin

memperhitungkan nilai sub-*fitness* dari antibodi, dengan kata lain untuk memilih bagian mana dari antibodi yang akan dijadikan vaksin, digunakan nilai *fitness* dari tiap fragmen dari antibodi, bukan nilai *fitness* antibodi tersebut secara keseluruhan. Untuk menghitung nilai *fitness* fragmen, dapat menggunakan persamaan (4.8) berikut,

$$fitness = \frac{1}{penalti_{protein} + 1} + \frac{1}{penalti_{lemak} + 1} + \frac{1}{penalti_{karbohidrat} + 1} + \frac{1000}{penalti_{biaya} + 1} \quad \textbf{(4.8)}$$

Proses ekstraksi vaksin dapat diilustrasikan dengan diagram alir



Gambar 4.7 Diagram alir ekstraksi vaksin

4.2.4.1 Ekstraksi M1

Pemilihan vaksin M1 mengacu pada *fitness* tiap fragmen. Nilai *fitness* tiap fragmen dapat dihitung dengan menggunakan persamaan (4.8), dimana tiap penalti dihitung dalam satu fragmen (satu kali makan).

Langkah langkah ekstraksi M1 dapat diuraikan sebagai berikut,

1. Ambil satu fragmen pada antibodi terpilih dengan metode roulette wheel.

fragmen	*fitness*	*probability*	rw
1	0,020230784	0,060244477	0,060244477
2	0,069093283	0,205750237	0,265994714
3	0,036834407	0,109687767	0,375682481
4	0,137914163	0,410689296	0,786371776
5	0,02633362	0,078417877	0,864789654
6	0,045405181	0,135210346	1
total	0,335811437		

r	0,465930526
terpilih	4

Gambar 4.8 Ilustrasi pemilihan fragmen M1

2. Pasangkan fragmen terbaik tersebut dengan satu fragmen lainnya, dan hitung nilai *fitness* masing-masing pasangan dengan persamaan (4.8), dengan nilai penalti merupakan jumlah nilai penalti dari pasangan tersebut.

Pasangan	41	42	43	45	46
Fitness	0,011808698	0,047248288	0,010672927	0,01720222	0,011826837

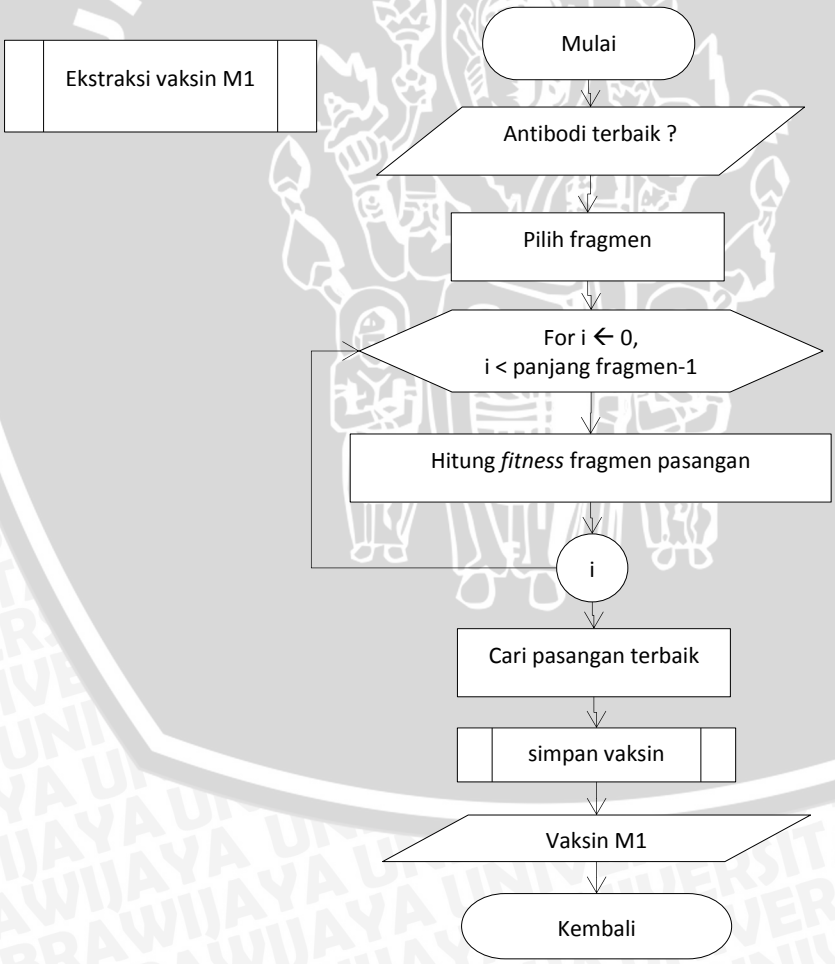
Gambar 4.9 Ilustrasi pasangan M1

3. Ambil satu pasangan terbaik sebagai vaksin M1, simpan ke dalam memori vaksin

				2	10	1	1					1	5	7	4				

Gambar 4.10 Ilustrasi vaksin M1

Diagram alir untuk ekstraksi M1 dapat dilihat pada Gambar 4.11



Gambar 4.11 diagram alir ekstraksi vaksin M1

4.2.4.2 Ekstraksi M2

Pemilihan fragmen untuk vaksin M2 sedikit berbeda. Pada M2, kandidat pasangan yang akan dipilih merupakan pasangan fragmen berurutan dengan tiap pasangan memiliki panjang B_l fragmen, yang dapat dicari dengan persamaan (4.9) berikut,

$$B_l = 2 + l$$

$$l = \begin{cases} 0, & r \leq 0.6 \\ 1, & 0.6 < r \leq 0.85 \\ 2, & 0.85 < r \leq 0.95 \\ 3, & 0.95 < r \end{cases} \quad \textbf{(4.9)}$$

Langkah langkah ekstraksi tahap kedua dapat dijabarkan sebagai berikut,

1. Cari nilai B_l dengan persamaan (4.9)

r	0,752131727
l	1
B_l	2+1 = 3

Gambar 4.12 Ilustrasi mencari nilai B_l

2. Pasangkan tiap fragmen dengan panjang pasangan B_l, dan hitung *fitness*-nya dengan persamaan (4.8) dengan nilai penalti merupakan jumlah nilai penalti dari pasangan tersebut

Pasangan	123	234	345	456
Fitness	0,007183254	0,007859815	0,007191222	0,007356626

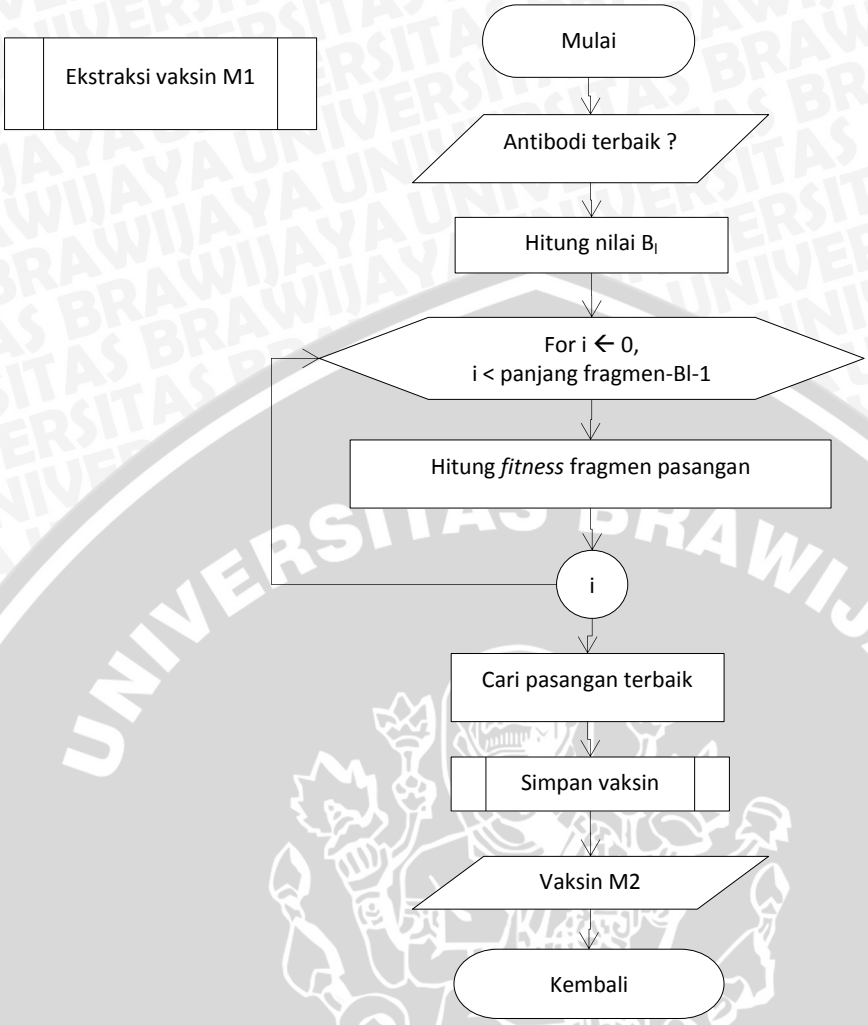
Gambar 4.13 Ilustrasi pasangan M2

3. Ambil satu pasangan terbaik sebagai vaksin M2, simpan ke dalam memori vaksin

				2	10	1	1	3	2	2	2	1	5	7	4			

Gambar 4.14 Ilustrasi vaksin M2

Diagram alir untuk proses ekstraksi M2 dapat dilihat pada Gambar 4.15.



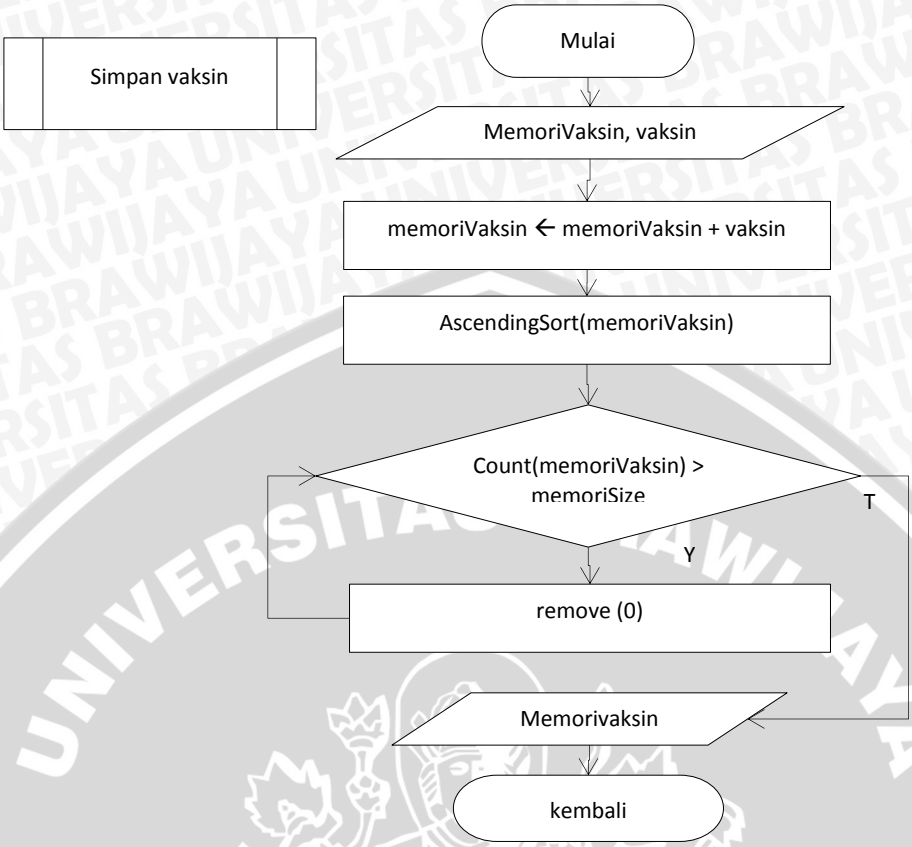
Gambar 4.15 Diagram alir ekstraksi vaksin M2

4.2.4.3 Proses penyimpanan vaksin

Setelah didapatkan vaksin utuh dari ekstraksi M1 dan M2, selanjutnya ialah menyimpan vaksin tersebut kedalam memori vaksin. Proses penyimpanan vaksin dapat dijabarkan sebagai berikut,

1. Jika memori vaksin masih kosong, atau belum mencapai batas memori vaksin, vaksin akan langsung disimpan
2. Jika memori vaksin sudah penuh, maka urutkan vaksin dalam memori vaksin berdasarkan *fitness*-nya, lalu buang vaksin yang memiliki *fitness* paling rendah.

Diagram alir dari proses penyimpanan vaksin dapat dilihat pada Gambar 4.16



Gambar 4.16 Diagram alir penyimpanan vaksin

4.2.5 Proses *crossover*

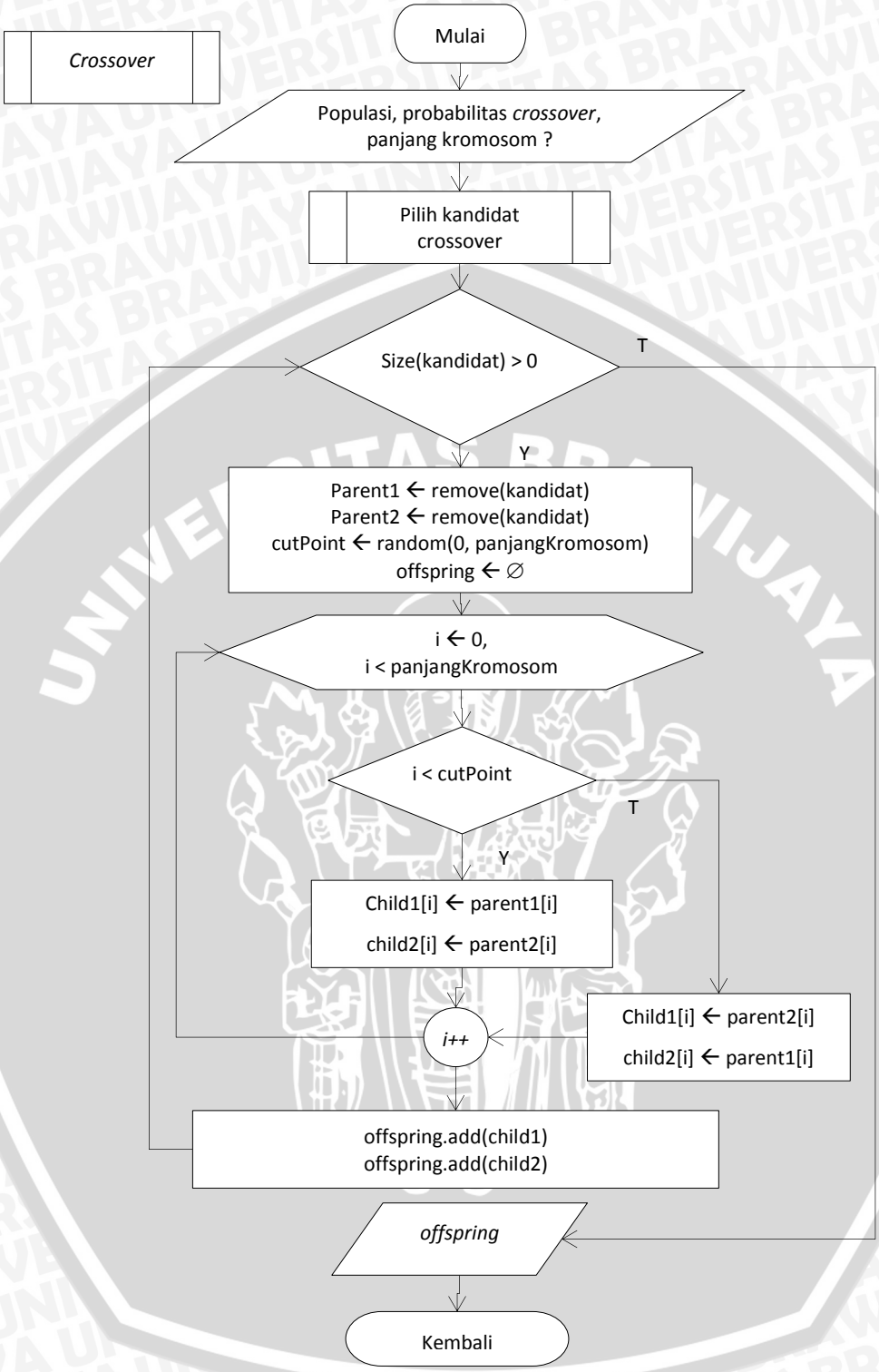
Metode *crossover* yang digunakan untuk penelitian ini ialah metode *one-point crossover*, yaitu memilih satu titik dalam kromosom sebagai *cut point*, lalu melakukan pertukaran gen antar kromosom mulai dari titik tersebut hingga sampai pada gen yang paling akhir.

Parent 1	1	10	13	3	10	15	4	8	11
Parent 2	4	10	13	4	7	13	4	9	14

offspring 1	1	10	13	3	10	15	4	9	14
offspring 2	4	10	13	4	7	13	4	8	11

Gambar 4.17 *One point crossover*

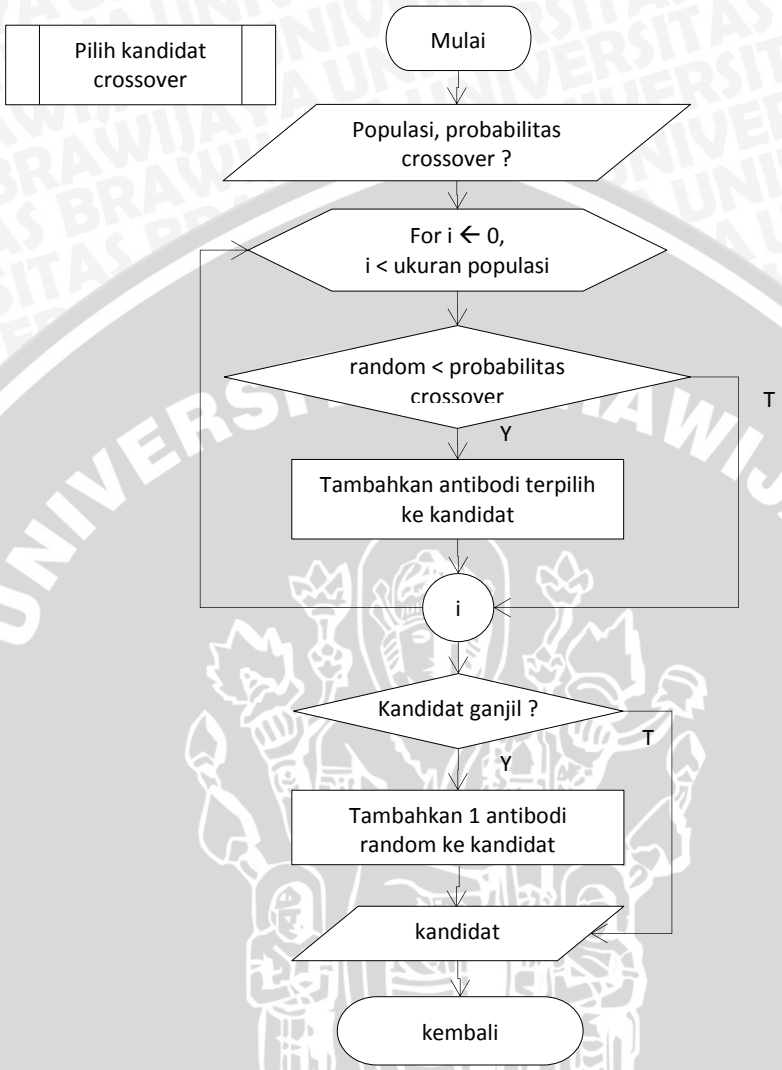
Terlihat pada Gambar 4.17, titik pemotongan berada pada gen ke 7, sehingga mulai dari gen ke-7 hingga gen terakhir akan ditukar dengan pasangannya untuk menghasilkan keturunan yang baru. Diagram alir proses *crossover* dapat dilihat pada Gambar 4.18.



Gambar 4.18 Diagram alir *crossover*

Proses pemilihan individu yang akan mengalami *crossover* dilakukan berdasarkan nilai probabilitas *crossover*, setiap antibodi akan mendapatkan angka acak dari 0 sampai 1, jika angka tersebut lebih kecil daripada nilai probabilitas *crossover* (*cr*), maka antibodi tersebut akan mengalami *crossover*. Jika kandidat *crossover* ternyata berjumlah ganjil, maka akan ditambah satu lagi

dengan cara mengambil random satu individu dalam populasi untuk mengalami *crossover*. Proses pemilihan individu dapat dilihat pada diagram alir pada Gambar 4.19



Gambar 4.19 Pemilihan antibodi untuk *crossover*

Setiap dua antibodi pada kandidat *crossover* akan dipasangkan menjadi *parent*1 dan *parent*2, keduanya lalu mengalami *crossover* untuk mendapatkan keturunan baru.

4.2.6 Proses *Mutasi*

Proses pemilihan antibodi yang akan mengalami mutasi hampir sama dengan proses pemilihan untuk *crossover*, hanya saja tidak ada proses pengecekan ganjil-genap seperti pada *crossover*. Setiap antibodi dalam populasi akan diberikan suatu nilai random dari 0 hingga 1, antibodi yang mendapatkan nilai yang lebih kecil daripada nilai probabilitas mutasi (*mr*) akan mengalami proses mutasi.

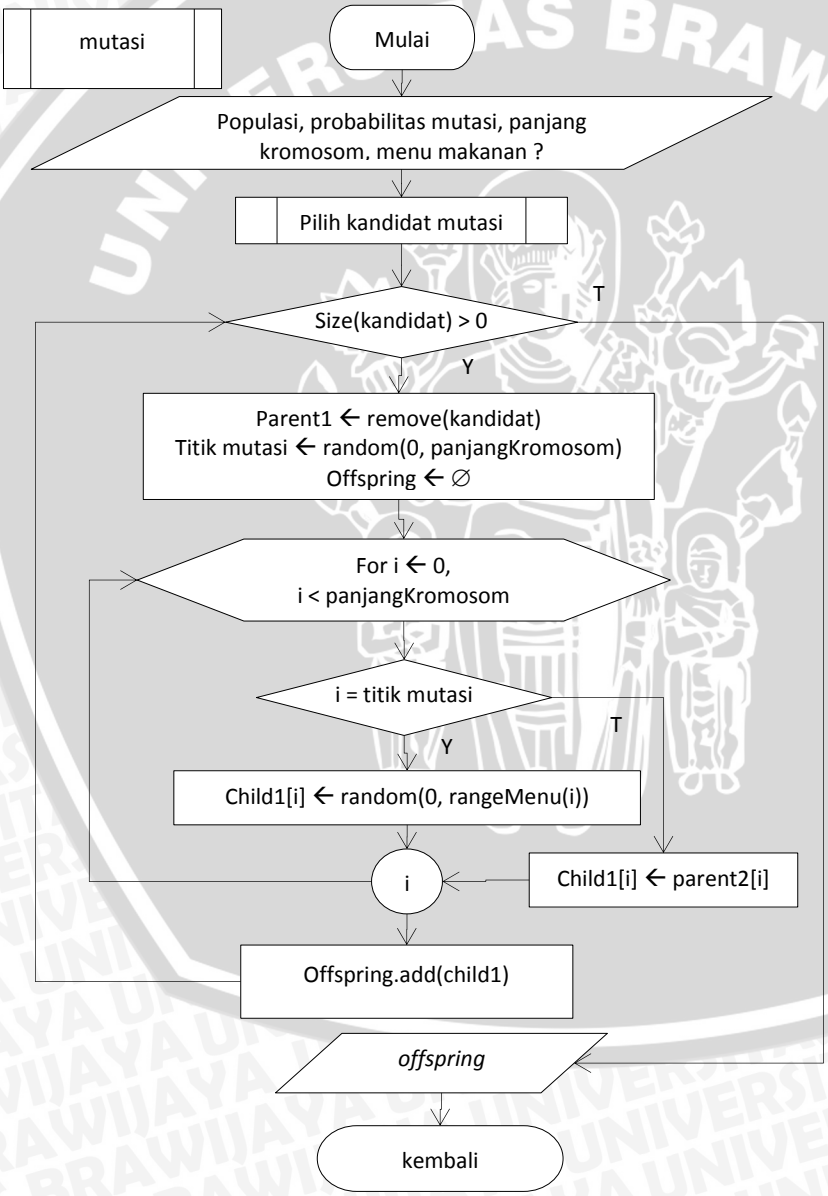
Metode mutasi yang dipilih untuk penelitian ini ialah *uniform mutation*. Proses mutasi dilakukan dengan memilih titik-titik mutasi, kemudian melakukan

perubahan gen pada titik-titik tersebut sesuai dengan *range* gen yang mungkin dapat masuk kedalam posisi tersebut. Seperti terlihat pada Gambar 4.20, titik terpilih untuk mutasi ialah pada gen ke 5 sehingga pada gen tersebut akan dilakukan mutasi dengan range yang sesuai. Misal pada gen ke 5 adalah gen untuk makanan pokok, maka mutasi yang terjadi berada dalam range sejumlah banyak data makanan pokok.

Parent	1	10	13	3	10	15	4	8	11
offspring	1	10	13	3	8	15	4	8	11

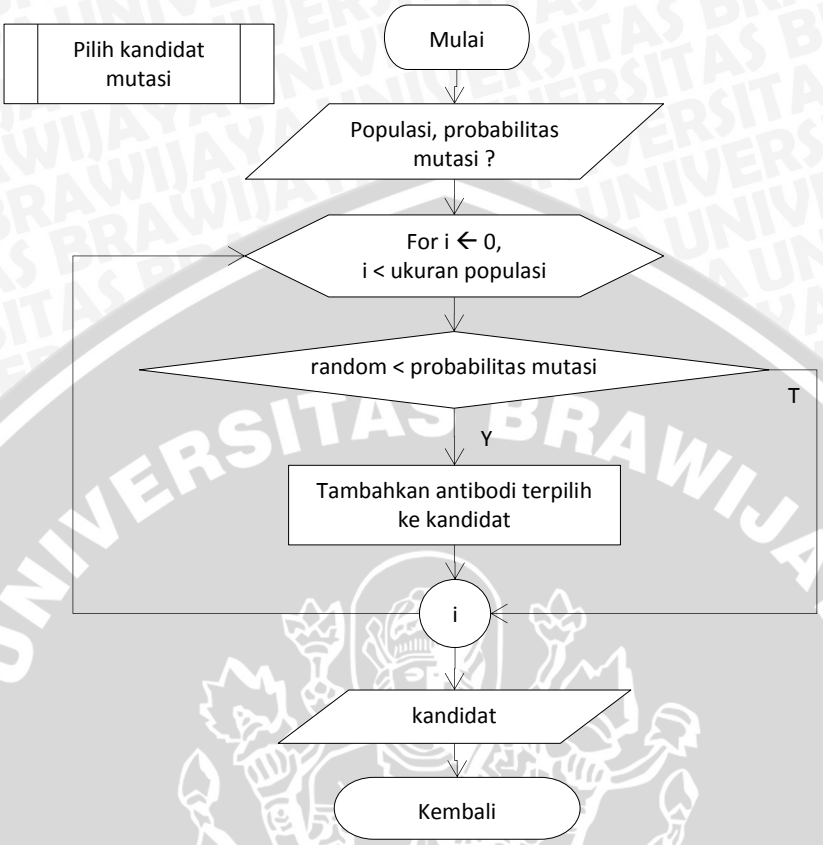
Gambar 4.20 *Uniform Mutation*

Diagram alir proses mutasi dapat dilihat pada Gambar 4.21.



Gambar 4.21 Diagram alir proses mutasi

Sedangkan proses pemilihan individu untuk mutasi dapat dilihat pada Gambar 4.22



Gambar 4.22 Diagram alir pemilihan kandidat mutasi

4.2.7 Vaksinasi

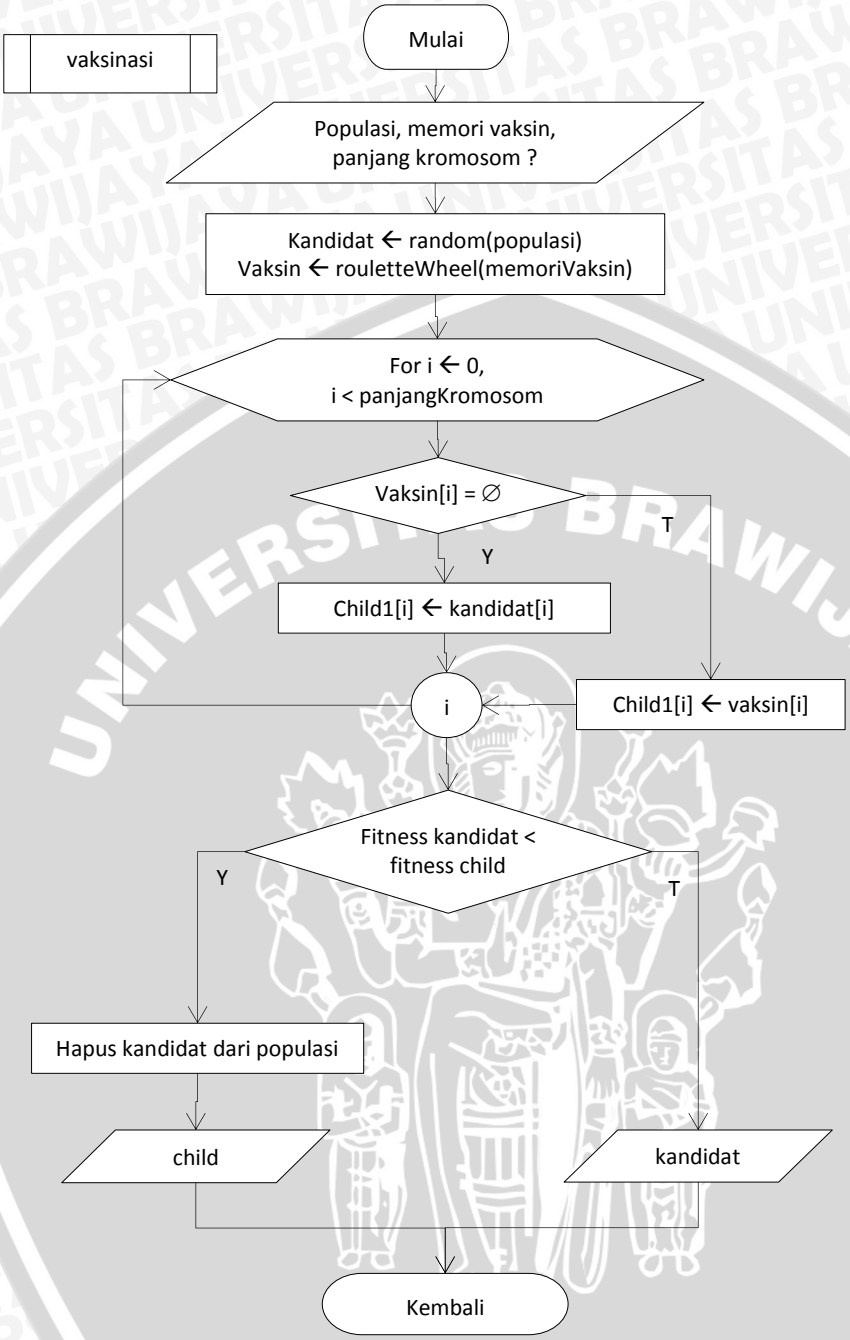
Antibodi yang akan divaksinasi dipilih secara random, begitupula dengan vaksin yang diambil dari *immune memory*. Antibodi terpilih akan diubah gennya sesuai dengan gen yang ada pada vaksin terpilih. Ilustrasi dari proses vaksinasi dapat dilihat pada Gambar 4.23.

Parent	1	10	13	3	10	15	4	8	11
Vaksin					7	13	2		

offspring	4	10	13	3	7	13	2	8	11

Gambar 4.23 Vaksinasi

Diagram alir proses vaksinasi dapat dilihat pada Gambar 4.24.



Gambar 4.24 Diagram alir proses vaksinasi

Untuk menjaga kualitas antibodi, jika hasil vaksinasi mengalami peningkatan nilai *fitness*, maka hasil vaksinasi akan menggantikan antibodi tersebut, sebaliknya jika nilai *fitness* hasil vaksinasi lebih rendah dari antibodi terpilih, maka hasil vaksinasi akan dibuang.

4.2.8 Seleksi Imun

Keberlangsungan hidup suatu antibodi ditentukan tak hanya oleh nilai *fitness* saja, tetapi juga memperhitungkan kesamaan antar antibodi yang ada pada populasi. Berikut langkah langkah untuk seleksi imun,

1. Mencari nilai kedekatan antar antibodi. Untuk mencari nilai kedekatan (*affinity*) antar antibodi, dapat digunakan persamaan (4.10)

$$A_{ij} = \frac{1}{1 + \sqrt{\sum_{v=1}^{m}(i_v - j_v)^2}} \quad \textbf{(4.10)}$$

Dimana A_{ij} merupakan nilai kedekatan antar antibodi *i* dan *j*, dan *m* merupakan banyak antibodi dalam satu populasi. Contoh perhitungan, pada 2 antibodi,

kromosom 1	3	8	8	1	3	4	1	1	3	10	7	4	...	2
kromosom 2	3	4	9	2	1	10	4	1	1	8	6	4	...	4

$$A_{12} = \frac{\mathbf{1}}{\mathbf{1} + \sqrt{(\mathbf{3}-\mathbf{3})^2 + (\mathbf{8}-\mathbf{4})^2 + (\mathbf{8}-\mathbf{9})^2 + (\mathbf{1}-\mathbf{2})^2 + \cdots + (\mathbf{2}-\mathbf{4})^2}}$$

= 0.067

2. Dari nilai kedekatan tersebut, dapat dicari nilai kesamaan (*similarity*). Nilai kesamaan dapat dihitung untuk tiap antibodi terhadap antibodi lainnya dimana nilai kedekatannya lebih besar daripada konstanta *similarity* (ϕ). Untuk mendapatkan nilai *similarity* digunakan persamaan (4.11)

$$C_i = \frac{banyak\ antibodi\ yang\ nilai\ kedekatannya\ terhadap\ antibodi\ i\ lebih\ besar\ daripada\ \phi}{N} \quad \textbf{(4.11)}$$

Dimana N merupakan banyak antibodi dalam satu populasi, dan ϕ ialah konstanta *similarity* . Semakin besar nilai C, semakin mirip antara antibodi satu dan lainnya dalam satu populasi.

3. Menghitung probabilitas seleksi. Dengan mengombinasikan antara nilai *fitness* antibodi dengan nilai kesamaan antibodi, sehingga didapatkan probabilitas seleksi yang tak hanya memperhitungkan nilai *fitness* semata, tetapi juga mempertimbangkan kesamaan antar antibodi dalam populasi. Probabilitas seleksi dapat dihitung dengan persamaan (4.12)

$$P = \mu P_f + (1-\mu)P_d \quad \textbf{(4.12)}$$

Dimana μ adalah konstanta *affinity*, Pf merupakan probabilitas berdasarkan nilai *fitness* yang dapat dicari dengan persamaan (4.13)

$$P_f = \frac{f_i}{\sum_{i=1}^{N} f_i} \quad \textbf{(4.13)}$$

Sedangkan Pd merupakan probabilitas berdasarkan nilai kesamaan yang dapat dicai dengan persamaan (4.14)

$$P_d = \frac{C_i}{\sum_{i=1}^{N} C_i} \quad \textbf{(4.14)}$$

Dimana N merupakan jumlah populasi, f_i adalah nilai *fitness* antibodi, dan C_i adalah nilai *similarity* antibodi.

4. Menghitung banyak antibodi yang akan bertahan ke generasi selanjutnya. Banyaknya antibodi yang akan bertahan ke generasi selanjutnya dapat dicari dengan menggunakan persamaan (4.15) berikut,

$$N_r = N - N_d = N(1 - \sigma PD) \quad \textbf{(4.15)}$$

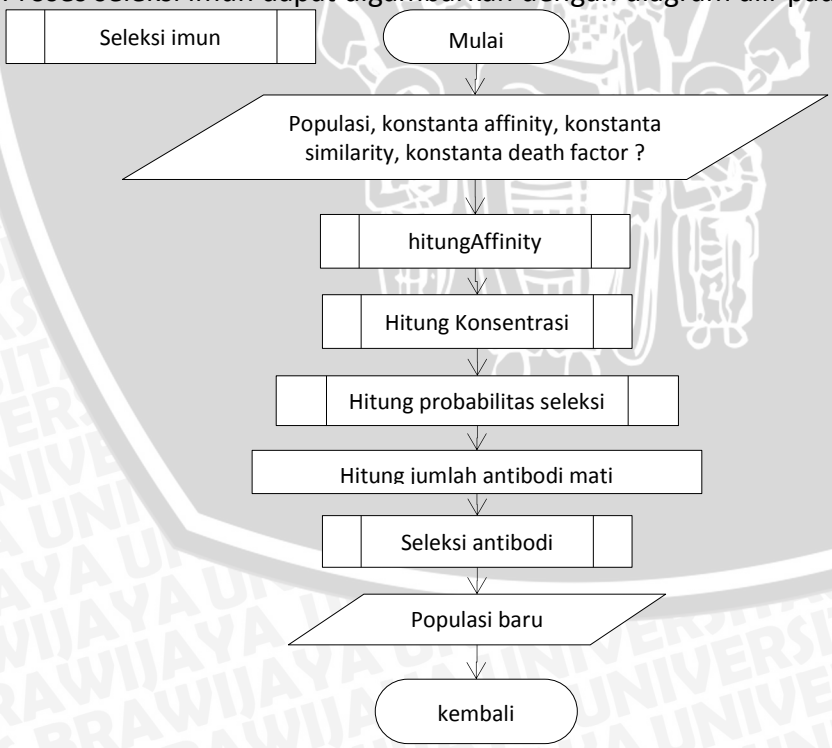
Dimana Nr ialah jumlah antibodi yang akan bertahan ke generasi selanjutnya, N ialah jumlah populasi saat ini, Nd adalah jumlah antibodi yang akan mati, σ adalah konstanta *death factor*, dan PD dapat dicari dengan persamaan (4.16)

$$PD = \frac{f_{avg}}{f_{max}} \quad \textbf{(4.16)}$$

Dimana f_{avg} merupakan rata rata nilai *fitness* dalam populasi, dan f_{max} merupakan nilai *fitness* tertinggi dalam populasi.

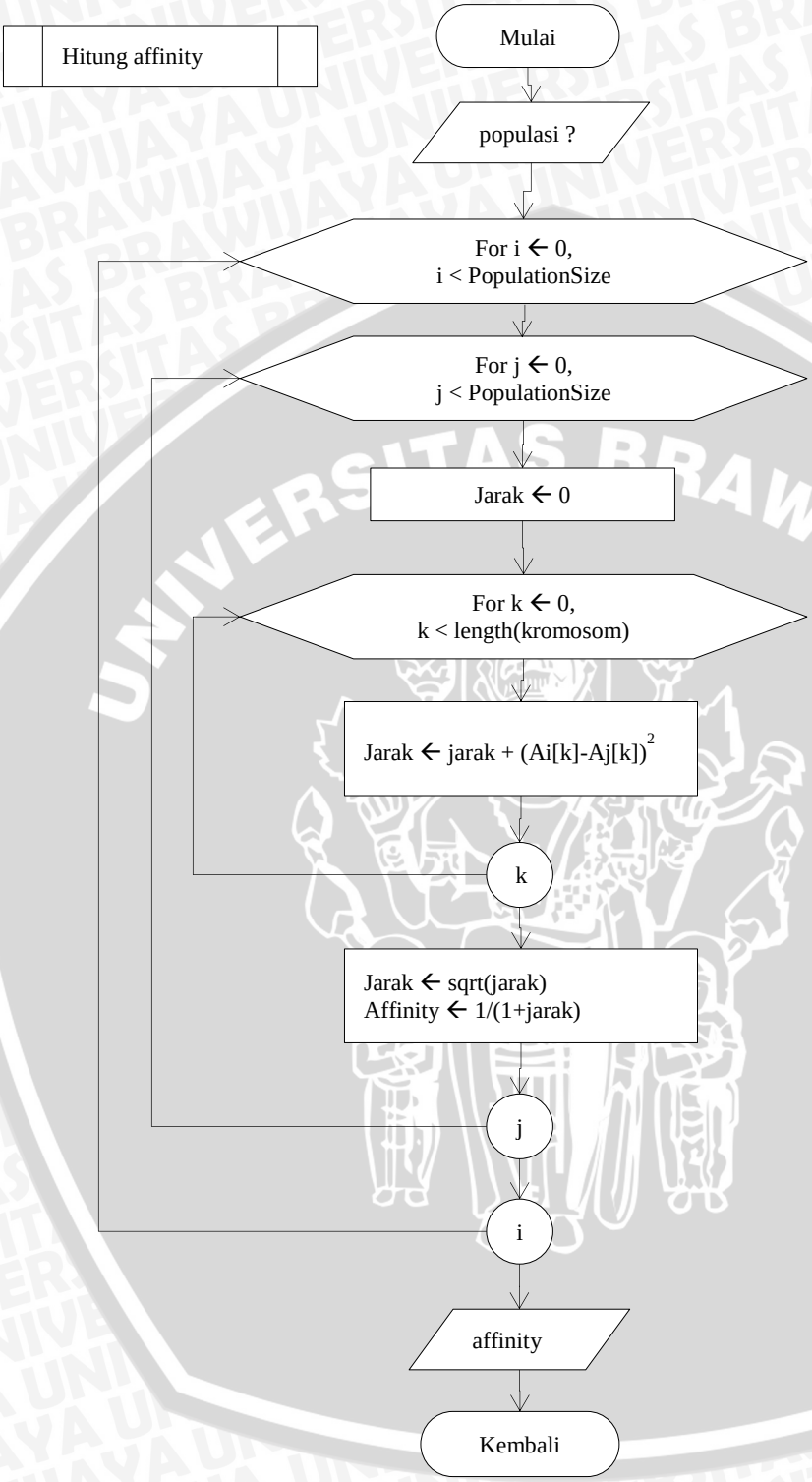
5. Proses seleksi dengan mengurutkan antibodi berdasarkan nilai probabilitas seleksi yang telah dihitung dengan menggunakan persamaan (4.12). Diambil sebanyak Nr antibodi yang memiliki probabilitas paling tinggi sebagai kandidat untuk maju ke generasi selanjutnya. Selain itu juga dilakukan proses pembersihan populasi dari antibodi kembar, serta generasi antibodi baru jika ternyata jumlah populasi turun hingga kurang dari nilai jumlah populasi awal.

Proses seleksi imun dapat digambarkan dengan diagram alir pada Gambar 4.25



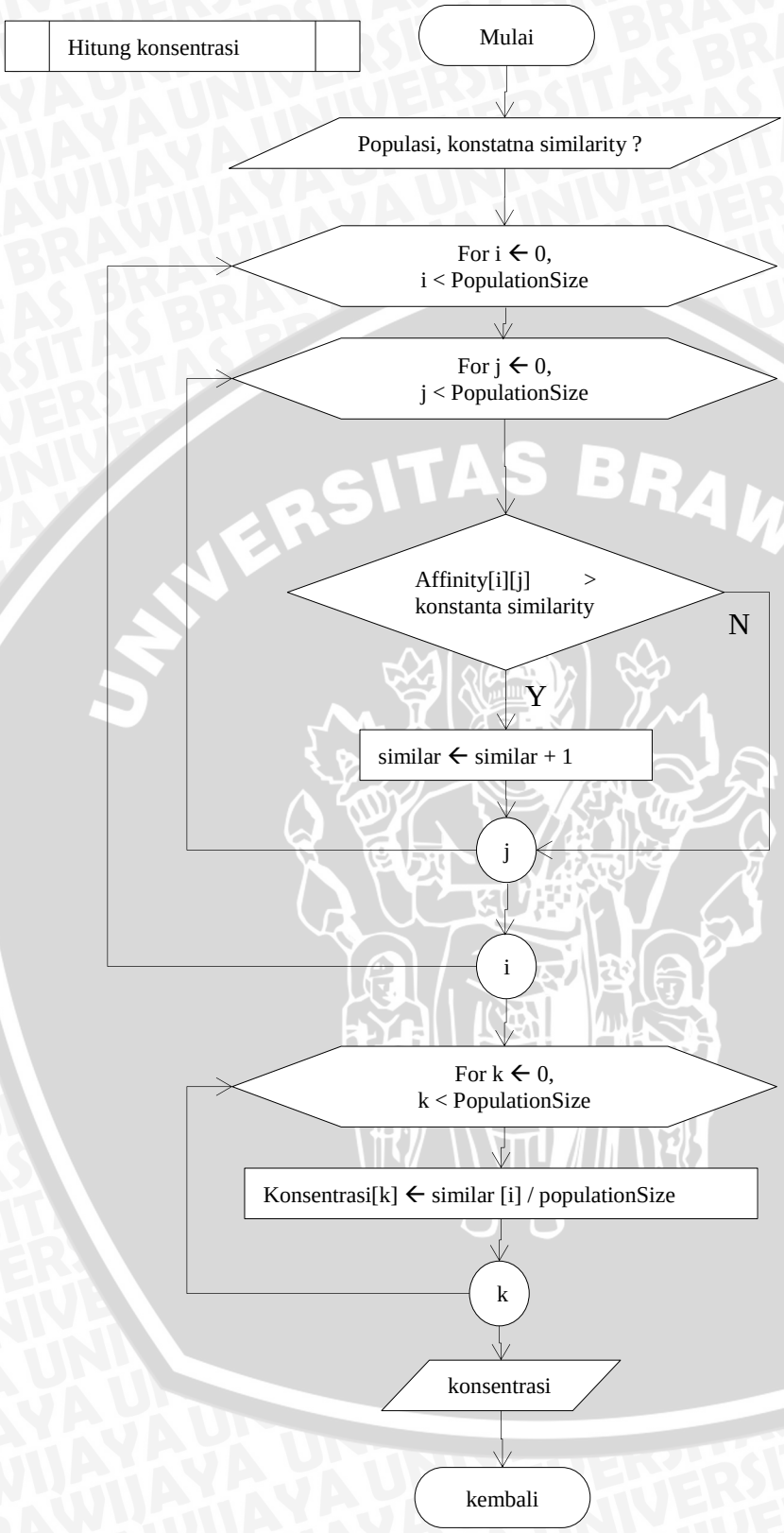
Gambar 4.25 Diagram alir proses seleksi imun

Proses hitung *affinity* dapat digambarkan dengan diagram alir Gambar 4.26



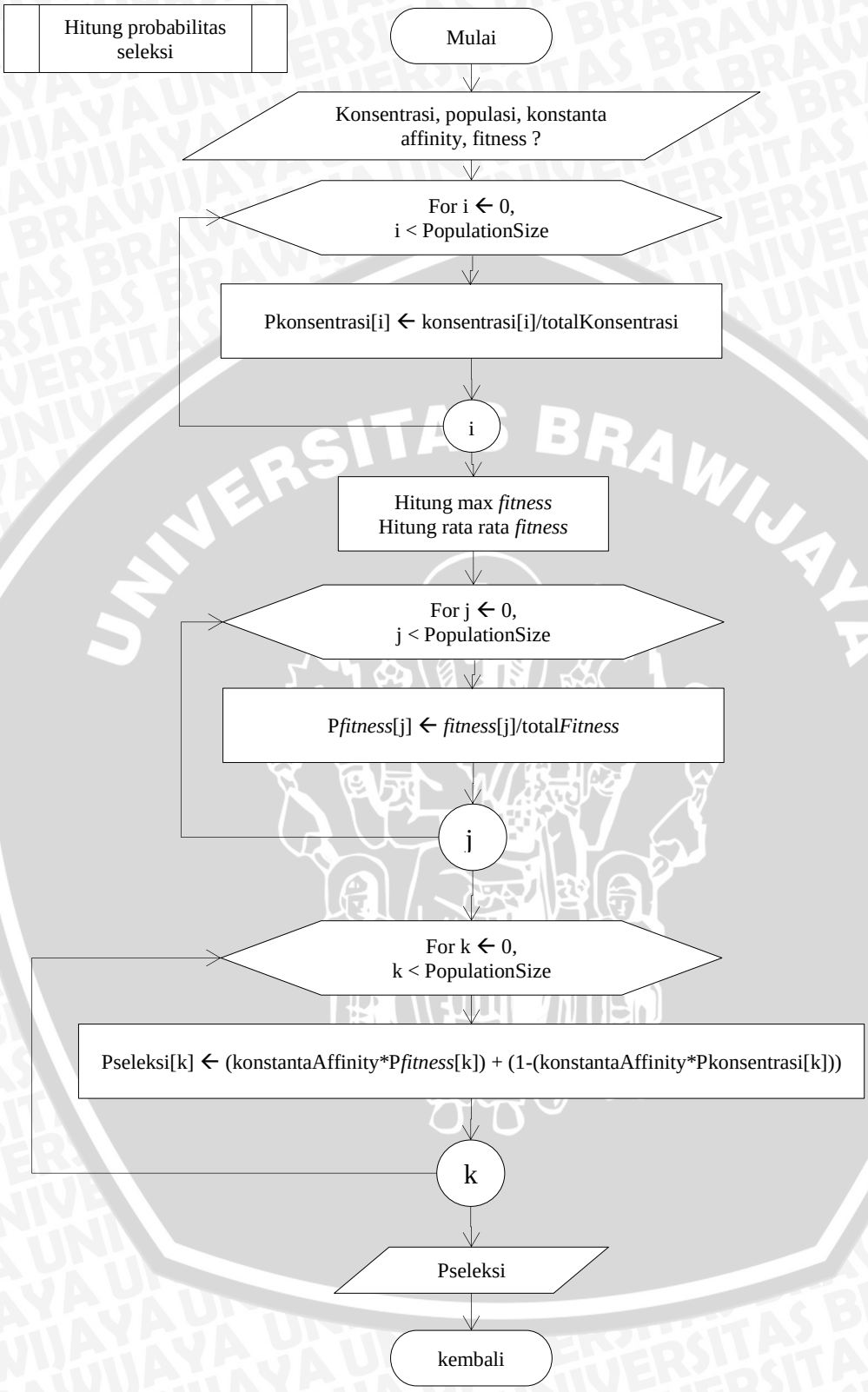
Gambar 4.26 Diagram alir proses hitung *affinity*

Proses hitung konsentrasi dapat dilihat pada Gambar 4.27



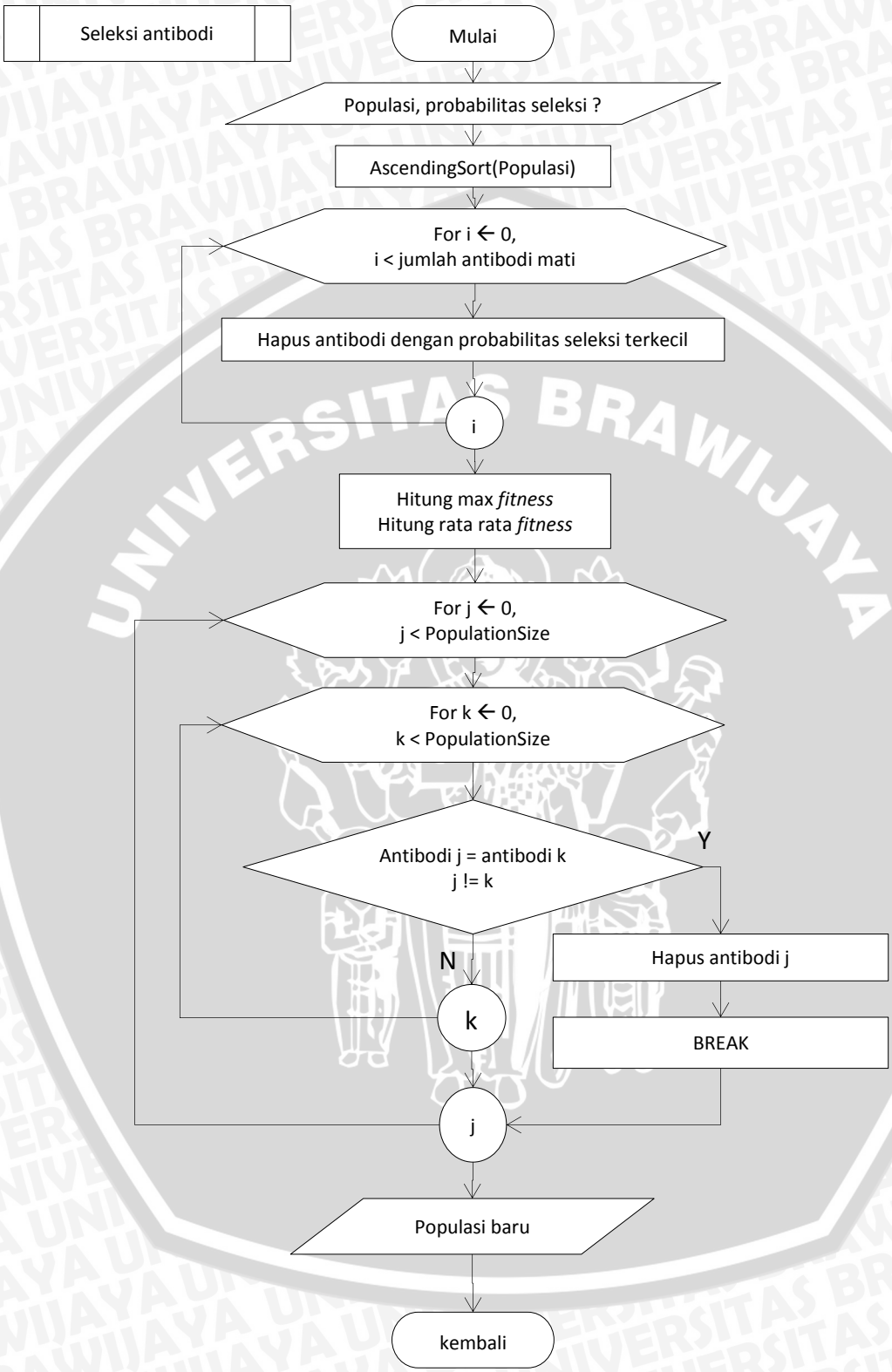
Gambar 4.27 Diagram alir proses hitung konsentasi

Untuk proses perhitungan probabilitas survival dapat dilihat pada Gambar 4.28



Gambar 4.28 Diagram alir proses perhitungan probabilitas seleksi

Untuk proses seleksi antibodi, dapat dilihat pada Gambar 4.29



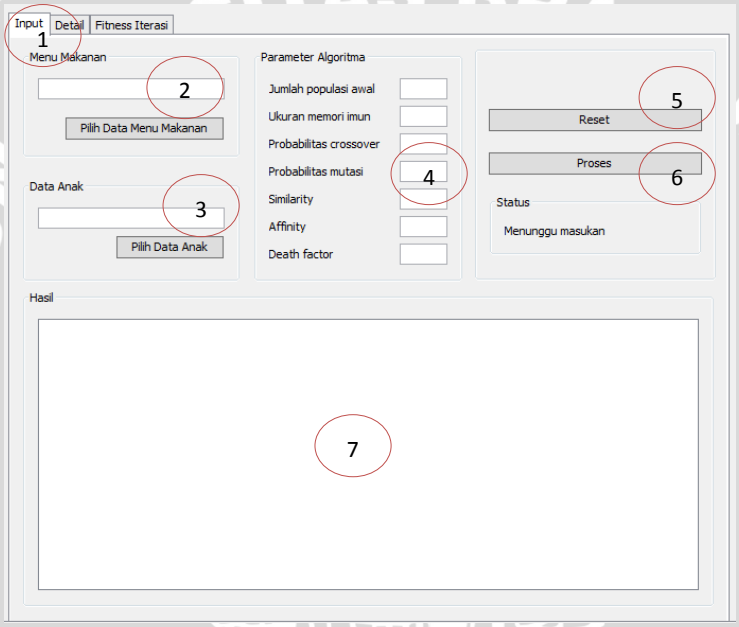
Gambar 4.29 Diagram alir proses seleksi antibodi

4.3 Perancangan Antarmuka

Antarmuka sistem dirancang dengan menggunakan perancang antarmuka SWING dari IDE Netbeans untuk Java.

4.3.1 Antarmuka Halaman Awal

Halaman awal difungsikan untuk menerima masukan dari pengguna. Masukan untuk sistem berupa kolom kolom teks untuk parameter algoritma, pencari berkas untuk dokumen Excel daftar menu makanan serta daftar anak panti asuhan, serta tombol untuk Reset atau mengatur ulang semua masukan kembali ke nilai awal, dan tombol proses untuk memulai proses algoritma setelah semua masukan terisi. Rancangan antarmuka halaman awal dapat dilihat pada gambar Gambar 4.30



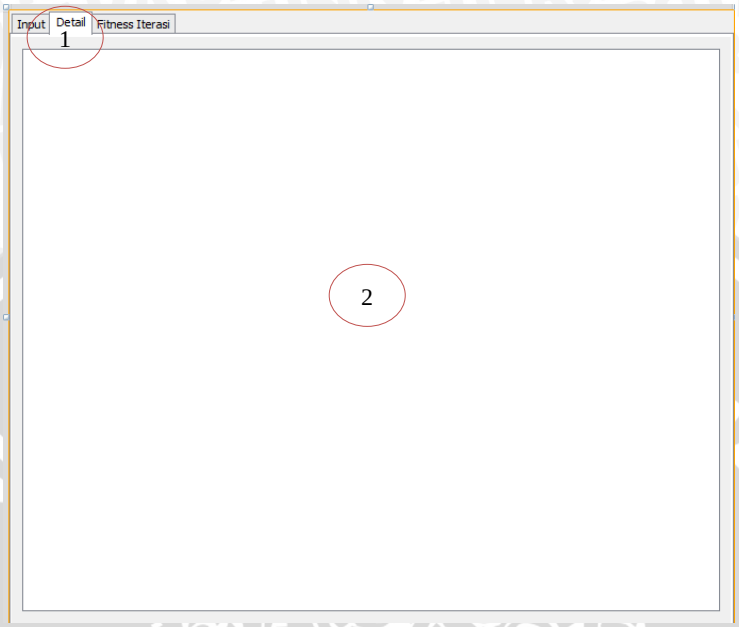
Gambar 4.30 Rancangan antarmuka halaman awal

Keterangan

1. Tab program yang aktif
2. Masukan data dari dokumen Excel untuk menu makanan
3. Masukan data dari dokumen Excel untuk anak panti asuhan
4. Masukan parameter algoritma
5. Tombol reset untuk mengembalikan masukan ke nilai semula
6. Tombol proses untuk memulai proses algoritma
7. Tampilan singkat hasil algoritma

4.3.2 Antarmuka Detail Hasil

Halaman detail hasil digunakan untuk menampilkan hasil algoritma secara mendetail. Rancangan antarmuka detail hasil dapat dilihat pada gambar Gambar 4.31



Gambar 4.31 Rancangan antarmuka detail hasil

Keterangan

1. Tab program yang aktif
2. Tampilan detail hasil algoritma yang berisi menu makanan tiap hari, kandungan gizi, kecukupan gizi, serta biaya yang dibutuhkan.

BAB 5 IMPLEMENTASI SISTEM

Bab ini menjelaskan tentang implementasi sistem berdasarkan analisis kebutuhan dan perancangan sistem yang telah dibuat.

5.1 Implementasi *Immune Genetic Algorithm*

Implementasi *Immune Genetic Algorithm* merupakan hasil perancangan sistem Optimasi Gizi pada Anak Panti Asuhan dengan *Immune Genetic Algorithm* ke dalam kode program.

5.1.1 Implementasi perhitungan kebutuhan gizi

Perhitungan kebutuhan gizi untuk menghitung kebutuhan gizi dari anak panti asuhan, baik kebutuhan untuk per hari maupun kebutuhan per makan.

Kode Program 5.1 Perhitungan kebutuhan gizi

```
1       static Nutrition[] getKebutuhanGiziMakan(String path) {
2
3           Nutrition[] result = new Nutrition[3]; // pagi siang malam
4
5           Workbook workbook;
6           Sheet sheetAnak = null;
7           boolean sukses = false;
8           try {
9               workbook = Workbook.getWorkbook(new File(path));
10              sheetAnak = workbook.getSheet(0);
11              sukses = true;
12          } catch (IOException ex) {

    Logger.getLogger(ConsoleUI.class.getName()).log(Level.SEVERE,  null,
13  ex);
14          } catch (BiffException ex) {

    Logger.getLogger(ConsoleUI.class.getName()).log(Level.SEVERE,  null,
15  ex);
16          }
17
18          Cell proteinCell;
19          Cell lemakCell;
20          Cell karboCell;
21          NumberCell nc;
22
23          double protein;
24          double lemak;
25          double karbo;
26          if (sukses) {
                for  (int  i  =  0,  row  =  13;  i  <  result.length;  i++,
27  row++) {
28                  proteinCell = sheetAnak.getCell(8, row);
29                  lemakCell = sheetAnak.getCell(9, row);
30                  karboCell = sheetAnak.getCell(10, row);
31
32                  nc = (NumberCell) proteinCell;
33                  protein = nc.getValue();
34
```

```
35                      nc = (NumberCell) lemakCell;
36                      lemak = nc.getValue();
37
38                      nc = (NumberCell) karboCell;
39                      karbo = nc.getValue();
40
41                      result[i] = new Nutrition(protein, lemak, karbo);
42
43                  }
44
45              }
46
47              return result;
48          }
```

Penjelasan Kode Program 5.1 tentang perhitungan kebutuhan gizi ialah sebagai berikut,

Baris 28 s.d. 41 : Pengambilan data kebutuhan gizi dari *file excel*.

5.1.2 Implementasi generasi populasi awal

Generasi populasi awal dilakukan dengan membuat antibodi antibodi baru sebanyak jumlah populasi awal yang diinginkan.

Kode Program 5.2 Generasi populasi awal

```
1  private void generateNewAntibody() {
2      Random r;
3      int gen;
4      int[] kromosom;
5
6      for (int i = population.size(); i < initPopulation; i++) {
7          r = new Random();
8          kromosom = new int[days * 12];
9          gen = 0;
10         for (int hari = 0; hari < days; hari++) {
               for (int waktuMakan = 0; waktuMakan < 3; waktuMakan++)
11 {
                   for (int jenisMakanan = 0; jenisMakanan < 4;
12 jenisMakanan++) {
                       kromosom[gen] =
13 r.nextInt(menuMakanan[jenisMakanan].getSize());
14                     gen++;
15                 }
16             }
17         }
18         population.add(new Antibody(kromosom, true));
19     }
20 }
21
```

Penjelasan Kode Program 5.2 tentang generasi populasi awal adalah sebagai berikut,

Baris 10 s.d. 12 : Perulangan untuk memastikan gen makanan sesuai dengan lokasinya pada kromosom.

Baris 13 : Proses pengisian kromosom dengan gen yang sesuai

Baris 18 : Proses pembuatan antibodi baru sesuai dengan kromosom yang telah dibuat sebelumnya.

5.1.3 Implementasi perhitungan nilai *fitness*

Implementasi perhitungan nilai *fitness* merupakan proses untuk menghitung nilai *fitness* antibodi yang merupakan tolak ukur seberapa baik antibodi tersebut terhadap antigen atau permasalahan.

Kode Program 5.3 Perhitungan nilai *fitness*

```
1  private void fitnessCalculation(Antibody antibody) {
2      double[] protein = new double[days * 12];
3      double[] lemak = new double[days * 12];
4      double[] karbohidrat = new double[days * 12];
5      double[] harga = new double[days * 12];
6
7      double[] selisihProtein = new double[days * 3];
8      double[] selisihLemak = new double[days * 3];
9      double[] selisihKarbohidrat = new double[days * 3];
10     int[] selisihHarga = new int[days * 3];
11
12     double[] selisihProteinPerHari = new double[days];
13     double[] selisihLemakPerHari = new double[days];
14     double[] selisihKarbohidratPerHari = new double[days];
15     int[] selisihHargaPerHari = new int[days];
16
17     double totalSelisihProtein = 0;
18     double totalSelisihLemak = 0;
19     double totalSelisihKarbohidrat = 0;
20     int totalSelisihHarga = 0;
21
22     double[] fragmenFitness = new double[days * 3];
23     double fitness;
24
25     int gen = 0;
26     int fragmen = 0;
27     int variasiKesamaan = 0;
28
29     for (int i = 0; i < selisihProtein.length; i++) {
30         selisihKarbohidrat[i] = 0;
```

```
31          selisihLemak[i] = 0;
32          selisihProtein[i] = 0;
33      }
34
35      for (int hari = 0; hari < days; hari++) {
36          selisihProteinPerHari[hari] = 0.0;
37          selisihLemakPerHari[hari] = 0.0;
38          selisihKarbohidratPerHari[hari] = 0.0;
39          selisihHargaPerHari[hari] = 0;
40
41          for (int waktuMakan = 0; waktuMakan < 3; waktuMakan++) {
42              selisihProtein[fragmen] = 0;
43              selisihLemak[fragmen] = 0;
44              selisihKarbohidrat[fragmen] = 0;
45              selisihHarga[fragmen] = 0;
                for (int jenisMakanan = 0; jenisMakanan < 4;
46  jenisMakanan++) {
                    protein[gen]        =        jumlahAnak        *
    menuMakanan[jenisMakanan].getMakanan(antibody.getGen(gen)).getKandu
47  nganGizi().getProtein();
                    lemak[gen]        =        jumlahAnak        *
    menuMakanan[jenisMakanan].getMakanan(antibody.getGen(gen)).getKandu
48  nganGizi().getFat();
                    karbohidrat[gen]        =        jumlahAnak        *
    menuMakanan[jenisMakanan].getMakanan(antibody.getGen(gen)).getKandu
49  nganGizi().getCarbohydrate();
                    harga[gen]        =        jumlahAnak        *
    menuMakanan[jenisMakanan].getMakanan(antibody.getGen(gen)).getHarga
50  ();
51
52                  selisihProtein[fragmen] += protein[gen];
53                  selisihLemak[fragmen] += lemak[gen];
54                  selisihKarbohidrat[fragmen] += karbohidrat[gen];
55                  selisihHarga[fragmen] += harga[gen];
56
57                  gen++;
58              }
59              selisihProteinPerHari[hari] += selisihProtein[fragmen];
60              selisihLemakPerHari[hari] += selisihLemak[fragmen];
                selisihKarbohidratPerHari[hari]                    +=
61  selisihKarbohidrat[fragmen];
62              selisihHargaPerHari[hari] += selisihHarga[fragmen];
63
                selisihHarga[fragmen]    =    selisihHarga[fragmen]    <=
64  avgCost ? 0 : selisihHarga[fragmen] - avgCost;
65
                if              (selisihProtein[fragmen]              <
66  kebutuhanGiziMakan[waktuMakan].getProtein() * 0.9
                        ||          selisihProtein[fragmen]           >
67  kebutuhanGiziMakan[waktuMakan].getProtein() * 1.1) {
                    selisihProtein[fragmen]                           =
68  Math.abs(kebutuhanGiziMakan[waktuMakan].getProtein()              -
```

```
      selisihProtein[fragmen]);
69                } else {
70                    selisihProtein[fragmen] = 0.0;
71                }
                  if                (selisihLemak[fragmen]                <
72  kebutuhanGiziMakan[waktuMakan].getProtein() * 0.9
                      ||            selisihLemak[fragmen]                 >
73  kebutuhanGiziMakan[waktuMakan].getProtein() * 1.1) {
                    selisihLemak[fragmen]                                   =
      Math.abs(kebutuhanGiziMakan[waktuMakan].getFat()                      -
74  selisihLemak[fragmen]);
75                } else {
76                    selisihLemak[fragmen] = 0.0;
77                }
                  if            (selisihKarbohidrat[fragmen]              <
78  kebutuhanGiziMakan[waktuMakan].getCarbohydrate() * 0.9
                      ||          selisihKarbohidrat[fragmen]             >
79  kebutuhanGiziMakan[waktuMakan].getCarbohydrate() * 1.1) {
                    selisihKarbohidrat[fragmen]                             =
      Math.abs(kebutuhanGiziMakan[waktuMakan].getCarbohydrate()             -
80  selisihKarbohidrat[fragmen]);
81                } else {
82                    selisihKarbohidrat[fragmen] = 0.0;
83                }
84
85                totalSelisihProtein += selisihProtein[fragmen];
86                totalSelisihLemak += selisihLemak[fragmen];
87                totalSelisihKarbohidrat += selisihKarbohidrat[fragmen];
88                totalSelisihHarga += selisihHarga[fragmen];
89
                  fragmen<i>Fitness</i>[fragmen]          =          1          /
      (selisihKarbohidrat[fragmen] + 1) + 1 / (selisihLemak[fragmen] + 1)
      + 1 / (selisihProtein[fragmen] + 1) + 1000 / (selisihHarga[fragmen]
90  + 1);
91                fragmen++;
92            }
93
              if              (selisihProteinPerHari[hari]                  <
94  kebutuhanPerHari.getProtein() * 0.9) {
                    selisihProteinPerHari[hari]                             =
95  kebutuhanPerHari.getProtein() - selisihProteinPerHari[hari];
              }       else      if      (selisihProteinPerHari[hari]        >
96  kebutuhanPerHari.getProtein() * 1.1) {
                    selisihProteinPerHari[hari]                             =
97  selisihProteinPerHari[hari] - kebutuhanPerHari.getProtein();
98            } else {
99                selisihProteinPerHari[hari] = 0.0;
100           }
              if (selisihLemakPerHari[hari] < kebutuhanPerHari.getFat() *
101 0.9) {
                    selisihLemakPerHari[hari] = kebutuhanPerHari.getFat() -
102 selisihLemakPerHari[hari];
              }       else      if      (selisihLemakPerHari[hari]        >
103 kebutuhanPerHari.getFat() * 1.1) {
```

```
            selisihLemakPerHari[hari] = selisihLemakPerHari[hari] -
104 kebutuhanPerHari.getFat();
105         } else {
106             selisihLemakPerHari[hari] = 0.0;
107         }
            if (selisihKarbohidratPerHari[hari] <
108 kebutuhanPerHari.getCarbohydrate() * 0.9) {
            selisihKarbohidratPerHari[hari] =
    kebutuhanPerHari.getCarbohydrate() -
109 selisihKarbohidratPerHari[hari];
        } else if (selisihKarbohidratPerHari[hari] >
110 kebutuhanPerHari.getCarbohydrate() * 1.1) {
            selisihKarbohidratPerHari[hari] =
    selisihKarbohidratPerHari[hari] -
111 kebutuhanPerHari.getCarbohydrate();
112         } else {
113             selisihKarbohidratPerHari[hari] = 0.0;
114         }
115         if (selisihHargaPerHari[hari] > avgCost * 3) {
            selisihHargaPerHari[hari] = selisihHargaPerHari[hari] -
116 (avgCost * 3);
117         } else {
118             selisihHargaPerHari[hari] = 0;
119         }
120     }
121
122     for (int i = 0; i < antibody.getKromosom().length; i++) {
123         for (int j = 0; j < antibody.getKromosom().length; j++) {
124             if (i % 4 == j % 4 && i != j) {
                variasiKesamaan += antibody.getKromosom()[i] ==
125 antibody.getKromosom()[j] ? 1 : 0;
126             }
127         }
128     }
129
130     double p = 0.0;
131     double l = 0.0;
132     double k = 0.0;
133     double h = 0.0;
134     for (int i = 0; i < selisihProteinPerHari.length; i++) {
135         p += selisihProteinPerHari[i];
136         l += selisihLemakPerHari[i];
137         k += selisihKarbohidratPerHari[i];
138         h += selisihHargaPerHari[i];
139     }
140
    fitness = 1 / (p + 1) + 1 / (l + 1) + 1 / (k + 1) + 1 /
141 (double) (variasiKesamaan + 1) + 1000 / (h + 1);
142
143     antibody.setFragmentFitness(fragmenFitness);
```

```
144         antibody.setFitness(fitness);
145         antibody.setSelisihProtein(selisihProtein);
146         antibody.setSelisihLemak(selisihLemak);
147         antibody.setSelisihKarbohidrat(selisihKarbohidrat);
148         antibody.setSelisihHarga(selisihHarga);
149         antibody.setSelisihProteinPerHari(selisihProteinPerHari);
150         antibody.setSelisihLemakPerHari(selisihLemakPerHari);
151     antibody.setSelisihKarbohidratPerHari(selisihKarbohidratPerHari);
152         antibody.setSelisihHargaPerHari(selisihHargaPerHari);
153
154     }
```

Penjelasan Kode Program 5.3 tentang perhitungan nilai *fitness* adalah sebagai berikut,

Baris 29 s.d. 33 : Proses inisialisasi selisih nilai gizi tiap kali makan

Baris 47 s.d. 50 : Proses pengambilan data nilai gizi dan harga untuk tiap makanan pada kromosom

Baris 52 s.d. 55 : Proses perhitungan jumlah nilai gizi dan harga tiap kali makan

Baris 59 s.d.62 : Proses perhitungan jumlah nilai gizi dan harga tiap hari

Baris 64 : Proses perhitungan selisih biaya makanan tiap kali makan dengan harapan biaya tiap kali makan

Baris 66 s.d. 83 : Proses perhitungan selisih jumlah gizi tiap kali makan dengan kebutuhan gizi tiap kali makan

Baris 90 : Proses perhitungan *sub-fitness* tiap kali makan

Baris 94 s.d. 114 : Proses perhitungan selisih jumlah gizi tiap hari dengan kebutuhan gizi tiap hari.

Baris 115 s.d. 119 : Proses perhitungan selisih biaya makan tiap hari dengan harapan biaya makan tiap hari

Baris 122 s.d. 128 : Proses perhitungan nilai variasi kesamaan menu makanan pada kromosom

Baris 134 s.d. 139 : Proses perhitungan jumlah selisih gizi dan biaya tiap hari

Baris 141 : Proses perhitungan nilai *fitness*

Baris 143 s.d. 152 : Proses pemasukan atribut *fitness*, serta selisih gizi dan harga pada antibodi

5.1.4 Implementasi ekstraksi vaksin

Implementasi ekstraksi vaksin merupakan proses untuk mengambil bagian kromosom individu untuk dijadikan vaksin.

Kode Program 5.4 Ekstraksi vaksin

```
1  private void ekstraksiVaksin() {
2
3      int maxFitness = 0;
4      Antibody antibodiTerpilih;
5      Antibody vaksin;
6
7      double[] pasanganFitness;
8      double pasanganProtein;
9      double pasanganLemak;
10     double pasanganKarbohidrat;
11     double pasanganHarga;
12     int bestPasangan;
13
14     for (int i = 0; i < population.size(); i++) {
           if (population.get(maxFitness).getFitness() <
15 population.get(i).getFitness()) {
16             maxFitness = i;
17         }
18     }
19     antibodiTerpilih = population.get(maxFitness);
20
21     // M1
       int fragmenTerpilih =
22 rouletteWheel(antibodiTerpilih.getFragmentFitness());
23
24     int[] kromosomVaksinM1 = new int[days * 12];
       pasanganFitness = new
25 double[antibodiTerpilih.getFragmentFitness().length];
26     pasanganProtein = 0;
27     pasanganLemak = 0;
28     pasanganKarbohidrat = 0;
29     pasanganHarga = 0;
30     bestPasangan = 0;
31
       for (int fragmen = 0; fragmen <
32 antibodiTerpilih.getFragmentFitness().length; fragmen++) {
           pasanganProtein =
       antibodiTerpilih.getSelisihProtein()[fragmenTerpilih] +
33 antibodiTerpilih.getSelisihProtein()[fragmen];
           pasanganLemak =
       antibodiTerpilih.getSelisihLemak()[fragmenTerpilih] +
34 antibodiTerpilih.getSelisihLemak()[fragmen];
           pasanganKarbohidrat =
       antibodiTerpilih.getSelisihKarbohidrat()[fragmenTerpilih] +
35 antibodiTerpilih.getSelisihKarbohidrat()[fragmen];
           pasanganHarga =
       antibodiTerpilih.getSelisihHarga()[fragmenTerpilih] +
36 antibodiTerpilih.getSelisihHarga()[fragmen];
37
```

```
38              if (fragmen != fragmenTerpilih) {
                    pasanganFitness[fragmen] = 1 / pasanganProtein + 1 /
39  pasanganLemak + 1 / pasanganKarbohidrat + 1000 / pasanganHarga;
40              } else {
41                  pasanganFitness[fragmen] = Double.NEGATIVE_INFINITY;
42              }
43          }
44
45          for (int i = 0; i < pasanganFitness.length; i++) {
46              if (pasanganFitness[bestPasangan] < pasanganFitness[i]) {
47                  bestPasangan = i;
48              }
49          }
50
51          for (int i = 0; i < kromosomVaksinM1.length; i++) {
52              if (i / 4 == fragmenTerpilih || i / 4 == bestPasangan) {
53                  kromosomVaksinM1[i] = antibodiTerpilih.getGen(i);
54              } else {
55                  kromosomVaksinM1[i] = -1;
56              }
57          }
58          vaksin = new Antibody(kromosomVaksinM1, false);
59          vaksin.setFitness(pasanganFitness[bestPasangan]);
60          addVaksin(vaksin);
61
62          //M2
63          Random r = new Random();
64          double random = r.nextDouble();
65          int m2Length = 2;
66          int[] kromosomVaksinM2 = new int[days * 12];
67
68          if (random < 0.6) {
69              m2Length += 0;
70          } else if (random < 0.85) {
71              m2Length += 1;
72          } else if (random < 0.9) {
73              m2Length += 2;
74          } else {
75              m2Length += 3;
76          }
77
78          if (m2Length > antibodiTerpilih.getFragmentFitness().length) {
79              m2Length = antibodiTerpilih.getFragmentFitness().length;
80          }
81
```

```
    pasanganFitness = new
double[antibodiTerpilih.getFragmentFitness().length - (m2Length -
1)];

    for (int pasangan = 0; pasangan < pasanganFitness.length;
pasangan++) {
        pasanganProtein = 0;
        pasanganLemak = 0;
        pasanganKarbohidrat = 0;
        pasanganHarga = 0;
        for (int i = 0; i < m2Length; i++) {
            pasanganProtein +=
antibodiTerpilih.getSelisihProtein()[pasangan + i];
            pasanganLemak +=
antibodiTerpilih.getSelisihLemak()[pasangan + i];
            pasanganKarbohidrat +=
antibodiTerpilih.getSelisihKarbohidrat()[pasangan + i];
            pasanganHarga +=
antibodiTerpilih.getSelisihHarga()[pasangan + i];
        }
        pasanganFitness[pasangan] = 1 / (pasanganProtein + 1) + 1 /
(pasanganLemak + 1) + 1 / (pasanganKarbohidrat + 1) + 1000 /
((double) pasanganHarga + 1);
    }

    bestPasangan = 0;
    for (int i = 0; i < pasanganFitness.length; i++) {
        if (pasanganFitness[bestPasangan] < pasanganFitness[i]) {
            bestPasangan = i;
        }
    }

    for (int i = 0; i < kromosomVaksinM2.length; i++) {
        if (i / 4 >= bestPasangan && i / 4 < bestPasangan +
m2Length) {
            kromosomVaksinM2[i] = antibodiTerpilih.getGen(i);
        } else {
            kromosomVaksinM2[i] = -1;
        }
    }

    vaksin = new Antibody(kromosomVaksinM2, false);
    vaksin.setFitness(pasanganFitness[bestPasangan]);
    addVaksin(vaksin);

}
```

Penjelasan Kode Program 5.4 tentang proses ekstraksi vaksin adalah sebagai berikut,

Baris 14 s.d. 19 : Proses pencarian antibodi terbaik untuk sumber vaksin

Baris 21 s.d. 60 : Proses pembuatan vaksin M1

Baris 62 s.d. 115 : proses pembuatan vaksin M2

Baris 22 : Proses pemilihan fragmen kandidat pertama vaksin M1

Baris 32 s.d. 43 : Proses pembuatan pasangan dan perhitungan *fitness* pasangan M1

Baris 45 s.d. 49 : Proses pencarian pasangan M1 terbaik

Baris 51 s.d. 57 : Proses penyimpanan gen pasangan M1

Baris 58 s.d. 60 : Proses penambahan vaksin M1 ke dalam memori vaksin.

Baris 68 s.d. 80 : Proses penentuan panjang vaksin M2

Baris 84 s.d. 96 : Proses pembuatan pasangan dan perhitungan *fitness* pasangan M2

Baris 98 s.d. 103 : Proses pencarian pasangan M2 terbaik

Baris 105 s.d. 111 : Proses penyimpanan gen pasangan M2

Baris 113 s.d. 115 : Proses penambahan vaksin M2 ke dalam memori vaksin

5.1.5 Implementasi proses *crossover*

Implementasi proses *crossover* merupakan proses untuk menyilangkan antara kromosom antibodi satu dengan lainnya untuk mendapatkan antibodi baru yang lebih baik.

Kode Program 5.5 Proses *crossover*

```
1  private ArrayList<Antibody> crossover() {
2      ArrayList<Antibody> kandidat = new ArrayList<>();
3      ArrayList<Antibody> offspring = new ArrayList<>();
4      Random r = new Random();
5      Antibody[] parent = new Antibody[2];
6      Antibody[] child = new Antibody[2];
7      int[][] childKromosom = new int[2][];
8      int cutPoint;
9
10     for (int i = 0; i < population.size(); i++) {
11         if (r.nextDouble() < crossoverRate) {
12             kandidat.add(population.get(i));
13         }
14     }
15
16     if (kandidat.size() % 2 != 0) {
17         kandidat.add(population.get(r.nextInt(population.size())));
18     }
```

```
19
20      while (!kandidat.isEmpty() && kandidat.size() % 2 == 0) {
21          parent[0] = kandidat.remove(0);
22          parent[1] = kandidat.remove(0);
23
24          cutPoint = r.nextInt(parent[0].getKromosom().length);
25          if (cutPoint <= 0) {
26              continue;
27          }
28
29          childKromosom[0] = new int[parent[0].getKromosom().length];
30          childKromosom[1] = new int[parent[0].getKromosom().length];
31
32          for (int i = 0; i < childKromosom[0].length; i++) {
33              if (i < cutPoint) {
34                  childKromosom[0][i] = parent[0].getKromosom()[i];
35                  childKromosom[1][i] = parent[1].getKromosom()[i];
36              } else {
37                  childKromosom[0][i] = parent[1].getKromosom()[i];
38                  childKromosom[1][i] = parent[0].getKromosom()[i];
39              }
40          }
41          child[0] = new Antibody(childKromosom[0], true);
42          child[1] = new Antibody(childKromosom[1], true);
43
44          offspring.add(child[0]);
45          offspring.add(child[1]);
46
47      }
48
49      return offspring;
50  }
```

Penjelasan Kode Program 5.5 tentang proses *crossover* adalah sebagai berikut,

Baris 10 s.d. 14 : Proses pemilihan kandidat antibodi untuk proses *crossover*

Baris 16 s.d. 18 : Proses pengecekan jumlah kandidat

Baris 20 : Perulangan untuk semua kandidat *crossover*

Baris 24 s.d. 27 : Proses pemilihan titik perpotongan

Baris 32 s.d. 40 : Proses persilangan antara dua *parent*

Baris 41 s.d. 42 : Proses penyimpanan hasil persilangan

Baris 44 s.d 45 : Proses penambahan hasil persilangan kedalam daftar *offspring*

5.1.6 Implementasi proses mutasi

Implementasi proses mutasi merupakan proses untuk mengubah secara random satu atau lebih gen pada kromosom antibodi dengan harapan untuk mendapatkan antibodi baru yang lebih baik.

Kode Program 5.6 Proses mutasi

```
1  private ArrayList<Antibody> mutation() {
2      ArrayList<Antibody> kandidat = new ArrayList<>();
3      ArrayList<Antibody> offspring = new ArrayList<>();
4
5      Random r = new Random();
6      Antibody parent;
7      Antibody child;
8      int[] childKromosom;
9      int mutationPoint;
10
11     for (int i = 0; i < population.size(); i++) {
12         if (r.nextDouble() < mutationRate) {
13             kandidat.add(population.get(i));
14         }
15     }
16
17     while (!kandidat.isEmpty()) {
18         parent = kandidat.remove(0);
19
20         childKromosom = new int[parent.getKromosom().length];
21         for (int mutate = 0; mutate < parent.getKromosom().length * 0.5; mutate++) {
22             mutationPoint = r.nextInt(parent.getKromosom().length);
23             for (int i = 0; i < childKromosom.length; i++) {
24                 if (i == mutationPoint) {
25                     childKromosom[i] = r.nextInt(menuMakanan[i % 4].getSize());
26                 } else {
27                     childKromosom[i] = parent.getKromosom()[i];
28                 }
29             }
30         }
31
32         child = new Antibody(childKromosom, true);
33         offspring.add(child);
34
35     }
```

```
36
37      return offspring;
38
39  }
```

Penjelasan Kode Program 5.6 tentang proses mutasi adalah sebagai berikut,

Baris 11 s.d. 14 : Proses pemilihan kandidat antibodi untuk proses mutasi

Baris 17 : Perulangan untuk tiap kandidat mutasi

Baris 21 : Perulangan untuk mutasi (berapa banyak mutasi dalam satu kromosom)

Baris 22 : Proses penentuan titik mutasi

Baris 23 s.d. 29 : Proses mutasi

Baris 32 : Proses penyimpanan hasil mutasi

Baris 33 : Proses penambahan hasil mutasi kedalam daftar *offspring*

5.1.7 Implementasi vaksinasi

Implementasi vaksinasi merupakan proses untuk mengubah beberapa gen pada kromosom antibodi sesuai dengan vaksin yang ada dengan harapan untuk mendapatkan antibodi baru yang lebih baik.

Kode Program 5.7 Vaksinasi

```
1   private void vaksinasi() {
2       double[] fitnessVaksin = new double[memoriVaksin.size()];
3       Random r = new Random();
4       int antibodiTerpilih = r.nextInt(population.size());
5       Antibody vaksin;
6       Antibody parent;
7       Antibody child;
8       int[] childKromosom;
9
10      for (int i = 0; i < memoriVaksin.size(); i++) {
11          fitnessVaksin[i] = memoriVaksin.get(i).getFitness();
12      }
13      vaksin = memoriVaksin.get(rouletteWheel(fitnessVaksin));
14      parent = population.get(antibodiTerpilih);
15      childKromosom = new int[parent.getKromosom().length];
16
17      for (int i = 0; i < childKromosom.length; i++) {
18          if (vaksin.getKromosom()[i] >= 0) {
19              childKromosom[i] = vaksin.getKromosom()[i];
20          } else {
```

```
21                childKromosom[i] = parent.getKromosom()[i];
22            }
23        }
24
25        child = new Antibody(childKromosom, parent.getIdAntibody());
26
27        fitnessCalculation(child);
28        fitnessCalculation(parent);
29        if (parent.getFitness() < child.getFitness()) {
30            population.remove(antibodiTerpilih);
31            population.add(child);
32        }
33    }
```

Penjelasan Kode Program 5.7 tentang vaksinasi adalah sebagai berikut,

Baris 10 s.d. 13 : Proses pemilihan vaksin

Baris 14 : Pemilihan antibodi yang akan divaksinasi

Baris 17 s.d. 23 : Proses vaksinasi

Baris 25 : Penyimpanan hasil vaksinasi

Baris 27 s.d. 28 : Perhitungan nilai *fitness* antibodi sebelum dan setelah divaksinasi

Baris 29 s.d. 32 : Seleksi hasil vaksinasi

5.1.8 Implementasi seleksi imun

Implementasi seleksi imun merupakan proses untuk menyeleksi antibodi dalam setiap iterasi untuk mendapatkan antibodi yang lebih baik untuk generasi selanjutnya

Kode Program 5.8 Seleksi imun

```
1   private void immuneSelection() {
        double[][] affinity = new
2   double[population.size()][population.size()];
3       double distance;
4
5       int[] similiarAntibody = new int[population.size()];
6       double[] konsentrasi = new double[population.size()];
7       double totalKonsentrasi = 0;
        double[] probabilitasKonsentrasi = new
8   double[population.size()];
9
10      double totalFitness = 0;
11      double rataFitness = 0;
12      double maxFitness = population.get(0).getFitness();
```

```
13      double[] probabilitas Fitness = new double[population.size()];
14
15      double[] probabilitasAkhir = new double[population.size()];
16
17      double pd;
18      int survive;
19      int mati;
20
21      for (int i = 0; i < similiar Antibody.length; i++) {
22          similiar Antibody[i] = 0;
23      }
24
25      for (int i = 0; i < population.size(); i++) {
26          for (int j = 0; j < population.size(); j++) {
27              distance = 0;
                for (int k = 0; k <
28  population.get(i).getKromosom().length; k++) {
                    distance += Math.pow((double)
    population.get(i).getKromosom()[k] - (double)
29  population.get(j).getKromosom()[k], 2);
30              }
31              distance = Math.sqrt(distance);
32              affinity[i][j] = 1 / (1 + distance);
33          }
34      }
35
36      for (int i = 0; i < similiar Antibody.length; i++) {
37          for (int j = 0; j < affinity[i].length; j++) {
                similiar Antibody[i] += affinity[i][j] >
38  simmiliarityConstant ? 1 : 0;
39          }
40      }
41
42      for (int i = 0; i < konsentrasi.length; i++) {
            konsentrasi[i] = (double) similiar Antibody[i] / (double)
43  population.size();
44          totalKonsentrasi += konsentrasi[i];
45      }
46      for (int i = 0; i < konsentrasi.length; i++) {
            probabilitasKonsentrasi[i] = konsentrasi[i] /
47  totalKonsentrasi;
48      }
49      for (int i = 0; i < population.size(); i++) {
50          total Fitness += population.get(i).get Fitness();
51          if (max Fitness < population.get(i).get Fitness()) {
52              max Fitness = population.get(i).get Fitness();
53          }
54      }
```

```
55      rata*Fitness* = total*Fitness* / population.size();
56
57      for (int i = 0; i < population.size(); i++) {
            probabilitas*Fitness*[i] = population.get(i).get*Fitness*() /
58  total*Fitness*;
59      }
60      for (int i = 0; i < population.size(); i++) {
            probabilitasAkhir[i]      =      (*affinity*Constant      *
    probabilitas*Fitness*[i])    +    ((1    -    *affinity*Constant)    *
61  probabilitasKonsentrasi[i]);
62      }
63
64      pd = rata*Fitness* / max*Fitness*;
65
66      mati = (int) Math.round(population.size() * deathFactor * pd);
67
68      if (population.size() > maxPopulation) {
69          mati = population.size() - maxPopulation;
70      }
71
72      survive = population.size() - mati;
73
74
    int min = 0;
75          ArrayList<Double>  probabilitasSeleksi  =  new  ArrayList
    <>();
76          for(int i = 0; i<probabilitasAkhir.length; i++){
77              probabilitasSeleksi.add(probabilitasAkhir[i]);
78          }
79          for (int i = 0; i < mati; i++) {
80              min = 0;
81              for(int j=0; j<probabilitasSeleksi.size(); j++){
82                  if(probabilitasSeleksi.get(min)                    >
    probabilitasSeleksi.get(j)){
83                      min = j;
84                  }
85              }
86              Antibody killed = population.remove(min);
87              probabilitasSeleksi.remove(min);
88
89          }
90
91      for (int i = 0; i < mati; i++) {
92          *Antibody* killed = population.remove(0);
93      }
94
95      boolean kembar;
96
```

```
97      Antibody a1, a2;
98      for (int i = 0; i < population.size(); i++) {
99          kembar = true;
100         a1 = population.get(i);
101         for (int j = 0; j < population.size(); j++) {
102             a2 = population.get(j);
103             if (i == j) {
104                 continue;
105             }
106             if (a1.getIdAntibody() == a2.getIdAntibody() && i != j) {
107                 population.remove(i);
108                 continue;
109             }
110             for (int k = 0; k < a1.getKromosom().length; k++) {
111                 if (a1.getKromosom()[k] != a2.getKromosom()[k]) {
112                     kembar = false;
113                     break;
114                 }
115             }
116             if (kembar) {
117                 population.remove(i);
118                 break;
119             }
120
121         }
122     }
123
124     generateNewAntibody();
125
126 }
```

Penjelasan Kode Program 5.8 tentang seleksi imun adalah sebagai berikut,

Baris 25 s.d. 34 : Perhitungan nilai *affinity* antar antibodi dalam populasi

Baris 36 s.d. 40 : Perhitungan nilai kemiripan antar antibodi

Baris 42 s.d. 45 : Perhitungan nilai konsentrasi kemiripan antibodi

Baris 46 s.d. 48 : Perhitungan nilai probabilitas berdasarkan konsentrasi kemiripan antibodi

Baris 49 s.d. 59 : Perhitungan nilai probabilitas berdasarkan *fitness* antibodi

Baris 60 s.d. 62 : Perhitungan nilai probabilitas akhir (probabilitas kelangsungan hidup) antibodi

Baris 64 : Perhitungan nilai PD

Baris 66 s.d. 70 : perhitungan jumlah antibodi yang akan mati

Baris 74 s.d. 94 : Proses penghapusan antibodi yang mati

Baris 96 s.d. 122 : Proses pengecekan antibodi kembar

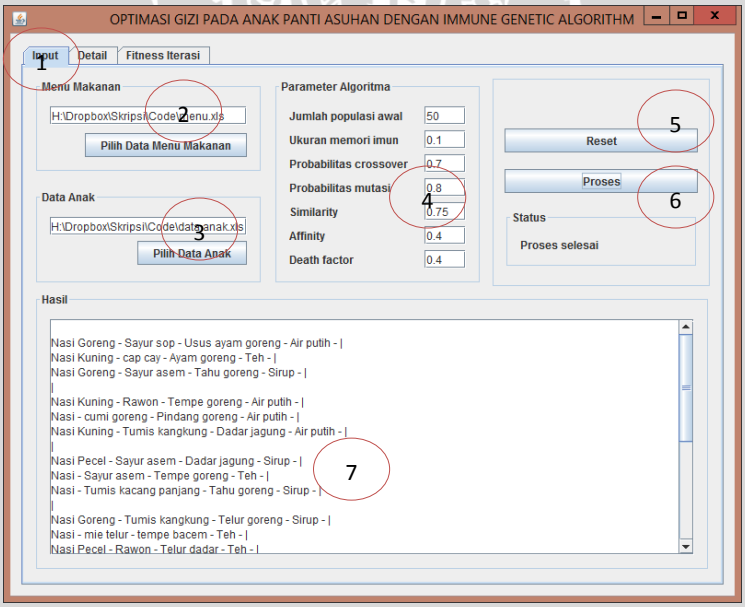
Baris 124 : Proses penambahan antibodi jika jumlah populasi kurang dari jumlah populasi awal

5.2 Implementasi antarmuka

Implementasi antarmuka merupakan hasil dari perancangan antarmuka yang telah dibuat sebelumnya.

5.2.1 Antarmuka Halaman Awal

Halaman awal difungsikan untuk menerima masukan dari pengguna. Masukan untuk sistem berupa kolom kolom teks untuk parameter algoritma, pencari berkas untuk dokumen Excel daftar menu makanan serta daftar anak panti asuhan, serta tombol untuk Reset atau mengatur ulang semua masukan kembali ke nilai awal, dan tombol proses untuk memulai proses algoritma setelah semua masukan terisi. Tampilan antarmuka halaman awal dapat dilihat pada gambar Gambar 5.1



Gambar 5.1 Antarmuka halaman awal

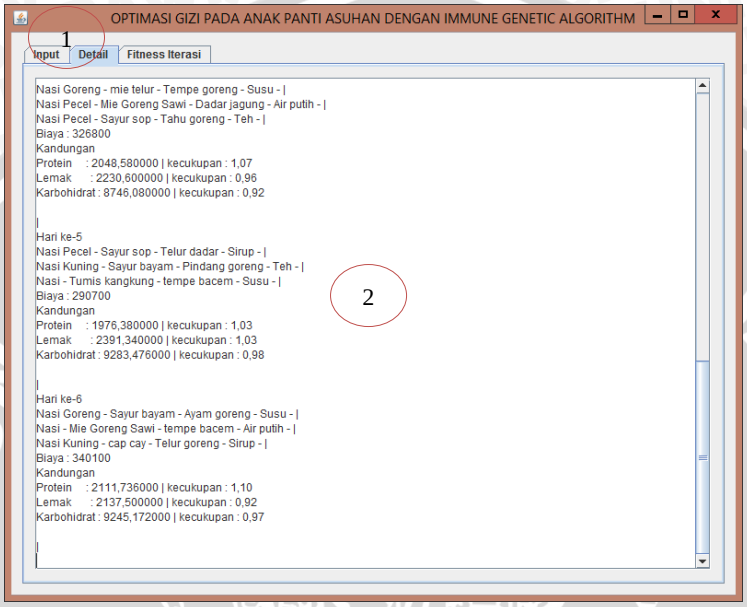
Keterangan

1. Tab program yang aktif
2. Masukan data dari dokumen Excel untuk menu makanan
3. Masukan data dari dokumen Excel untuk anak panti asuhan
4. Masukan parameter algoritma
5. Tombol reset untuk mengembalikan masukan ke nilai semula

6. Tombol proses untuk memulai proses algoritma
7. Tampilan singkat hasil algoritma

5.2.2 Antarmuka Detail Hasil

Halaman detail hasil digunakan untuk menampilkan hasil algoritma secara mendetail. Tampilan antarmuka detail hasil dapat dilihat pada gambar Gambar 5.2



Gambar 5.2 Antarmuka detail hasil

Keterangan

1. Tab program yang aktif
2. Tampilan detail hasil algoritma yang berisi menu makanan tiap hari, kandungan gizi, kecukupan gizi, serta biaya yang dibutuhkan.

BAB 6 PENGUJIAN DAN ANALISIS

Bab ini menjelaskan tentang pengujian dan analisis hasil pengujian dari sistem Optimasi Gizi pada Anak Panti Asuhan dengan *Immune Genetic Algorithm*. Pengujian yang dilakukan mengacu pada seksi 3.6 yang meliputi pengujian jumlah populasi awal, pengujian probabilitas *crossover*, pengujian probabilitas mutasi, pengujian konstanta *affinity*, pengujian konstanta *similarity*, serta pengujian konstanta *death factor*.

6.1 Pengujian jumlah populasi awal

Pengujian jumlah populasi awal bertujuan untuk mengetahui jumlah populasi awal yang paling optimal. Jumlah populasi awal yang diuji dimulai dari 10 hingga 100 dengan pengujian setiap kelipatan 10, dan maksimum jumlah populasi 100. Untuk mendapatkan jumlah populasi yang paling optimal, pengujian dilakukan masing-masing lima kali untuk tiap jumlah populasi awal, dengan nilai parameter lainnya ialah sebagai berikut,

1. Iterasi maksimum = 1000
2. Jumlah populasi awal = 10 s.d. 100, pengujian tiap kelipatan 10
3. probabilitas *crossover* = 0,6
4. probabilitas mutasi = 0,1
5. konstanta *affinity* = 0,3
6. konstanta *similarity* = 0,9
7. konstanta *death factor* = 0,5

Dengan parameter tersebut, hasil dari pengujian populasi awal dapat dilihat pada Tabel 6.1

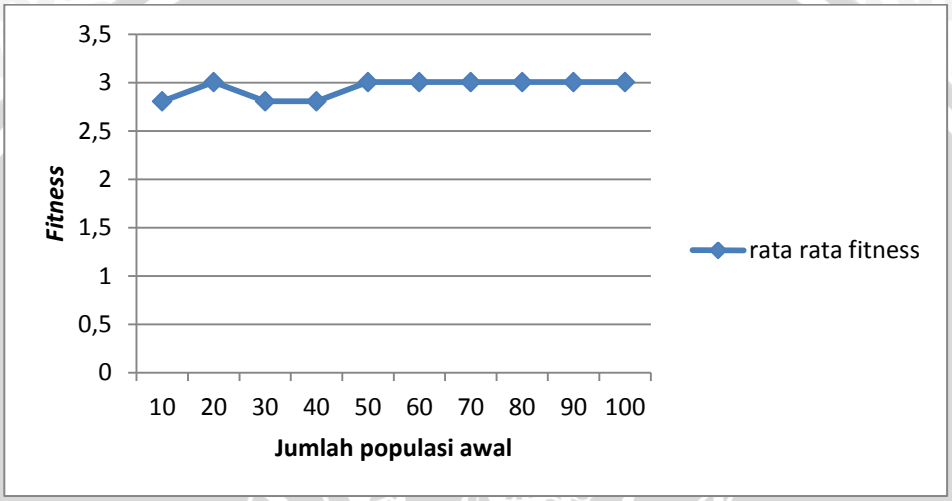
Tabel 6.1 Hasil pengujian populasi awal

Populasi		Uji coba ke-					Max iterasi	Rata iterasi	Min iterasi	Rata *fitness*
		1	2	3	4	5				
	k	584	710	808	857	888				
10	f	2,0064	3,0048	3,0048	3,0048	3,0048	888	769,4	584	2,8052
	k	580	832	712	723	649				
20	f	3,0048	3,0049	3,0049	3,0049	3,0048	832	699,2	580	3,0049
	k	476	313	808	408	642				
30	f	3,0049	3,0048	3,0049	3,0048	2,0066	808	529,4	313	2,8052
	k	662	414	297	362	430				
40	f	3,0048	3,0049	3,0049	3,0049	2,0066	662	433	297	2,8052
	k	132	276	282	452	511				
50	f	3,0048	3,0049	3,0049	3,0049	3,0049	511	330,6	132	3,0049
	k	154	378	258	229	174				
60	f	3,0049	3,0048	3,0049	3,0048	3,0049	378	238,6	154	3,0049
	k	242	144	274	375	130				
70	f	3,0049	3,0049	3,0049	3,0048	3,0049	375	233	130	3,0049

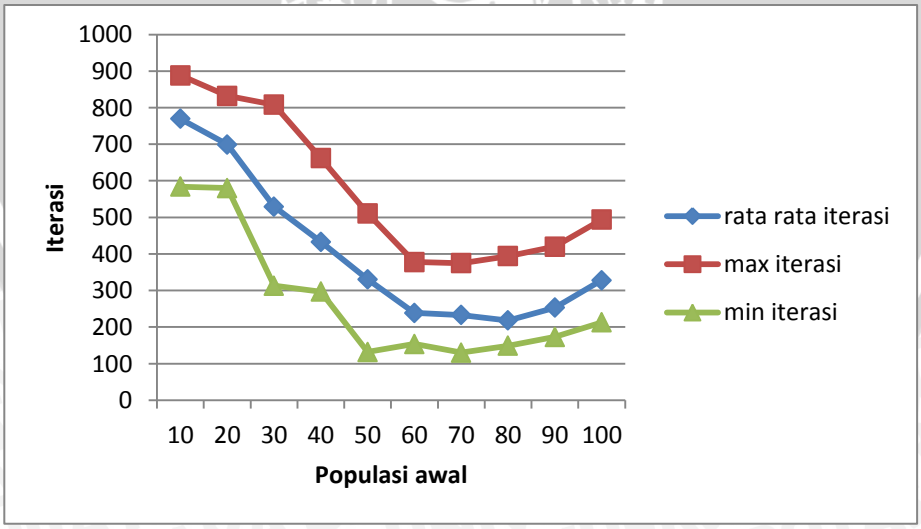
	k	181	164	149	203	394				
80	f	3,0049	3,0049	3,0049	3,0049	3,0048	394	218,2	149	3,0049
	k	189	420	271	173	212				
90	f	3,0049	3,0049	3,0049	3,0049	3,0049	420	253	173	3,0049
	k	494	213	334	313	285				
100	f	3,0049	3,005	3,0049	3,0049	3,0048	494	327,8	213	3,0049

Dengan k = konvergen pada iterasi ke- , dan f = nilai *fitness* saat konvergen

Dari hasil pengujian tersebut dapat dibuat grafik hasil pengujian yang ditampilkan pada Gambar 6.1 untuk hasil *fitness* dan Gambar 6.2 untuk hasil kecepatan konvergensi.



Gambar 6.1 Grafik *fitness* hasil pengujian jumlah populasi awal



Gambar 6.2 Grafik konvergensi hasil pengujian jumlah populasi awal

Hasil pengujian ini merupakan hasil pengujian yang telah diberi pembatas jumlah maksimum populasi. Apabila tidak diberi pembatas jumlah maksimum populasi, paka populasi dapat mengembang hingga jumlah yang sangat besar

hingga mengakibatkan program tidak dapat berjalan dengan baik. Pembatas jumlah maksimum populasi pada pengujian ini ialah maksimum 100 antibodi dalam satu populasi.

Dari grafik pertama (grafik *fitness*), dapat dilihat terdapat sedikit variasi pada setiap jumlah populasi. Untuk nilai *fitness* terlihat nilai *fitness* lebih stabil pada jumlah populasi lebih dari 50 yang stabil pada angka 3. Dari grafik kedua (grafik kecepatan konvergensi), terlihat semakin banyak jumlah populasi cenderung berbanding lurus dengan kecepatan konvergensi, dengan kata lain semakin banyak jumlah populasi, konvergensi akan semakin cepat. Konvergensi mulai stabil pada jumlah populasi awal 60 hingga 80, lebih dari itu konvergensi cenderung kembali meningkat meskipun tidak setinggi jumlah populasi yang kurang dari 60.

Fenomena tersebut dapat disebabkan karena jumlah populasi pada setiap iterasi dapat bervariasi. Pada rentang jumlah menengah, jumlah populasi dapat bertambah hingga jumlah maksimum populasi, serta tetap mempertahankan beberapa antibodi terbaik dari generasi sebelumnya dan memiliki cukup antibodi untuk mencari antibodi terbaik dengan relatif cepat. Nilai populasi awal yang terlalu besar dapat mengakibatkan sulitnya mendapatkan variasi baru dari antibodi karena jumlah populasi akan berusaha kembali ke batas populasi maksimum saat jumlah populasi melebihi nilai tersebut, sedangkan jumlah populasi awal yang terlalu kecil mengakibatkan sulitnya mencari antibodi yang memiliki *fitness* yang baik karena terbatasnya kombinasi gen dalam populasi.

Dari pengujian jumlah populasi awal, dapat disimpulkan jika parameter terbaik untuk jumlah populasi awal berada di rentang menengah atas, yaitu 70, karena menghasilkan *fitness* yang baik dan kecepatan konvergensi yang relatif cepat.

6.2 Pengujian probabilitas *crossover*

Pengujian probabilitas *crossover* bertujuan untuk mengetahui nilai probabilitasi *crossover* yang paling optimal. Nilai probabilitas *crossover* akan diuji coba dari nilai 0,0 hingga 1,0 dengan penujian pada setiap kelipatan 0,1. Untuk mendapatkan nilai probabilitas *crossover* yang paling optimal, pengujian dilakukan masing-masing lima kali untuk tiap nilai probabilitas dan akan mencatat hasil nilai *fitness* dan pada iterasi keberapa terjadi konvergensi, dengan nilai parameter lainnya ialah sebagai berikut,

1. Banyak iterasi = 1000
2. Jumlah populasi awal = 70
3. probabilitas *crossover* = 0,0 s.d. 1,0 , dengan pengujian setiap kelipatan 0,1
4. probabilitas mutasi = 0,1
5. konstanta *affinity* = 0,3
6. konstanta *similarity* = 0,9
7. konstanta *death factor* = 0,5

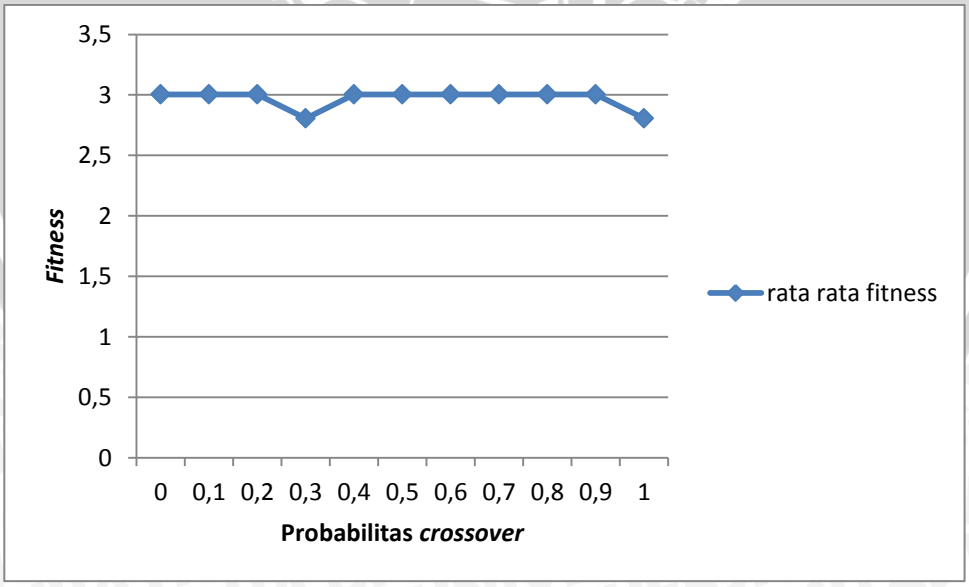
Hasil pengujian ditampilkan pada Tabel 6.2

Tabel 6.2 Hasil pengujian probabilitas *crossover*

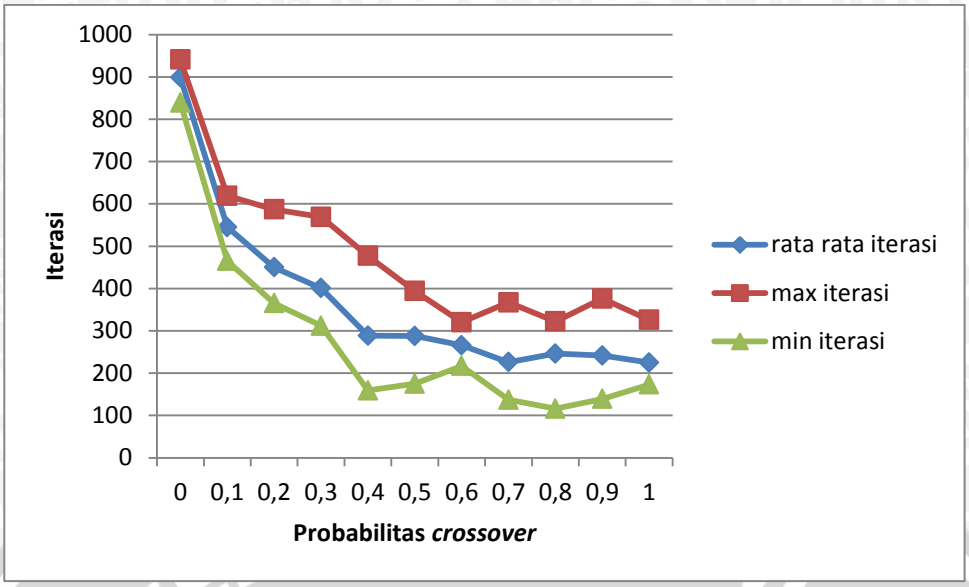
Probabilitas *crossover*		Uji coba ke-					Max iterasi	Rata iterasi	Min iterasi	Rata *fitness*
		1	2	3	4	5				
0	k	934	839	931	941	849	941	898,8	839	3,00474
	f	3,0048	3,0047	3,0046	3,0048	3,0048				
0,1	k	465	605	619	512	523	619	544,8	465	3,0049
	f	3,0049	3,0049	3,0049	3,0049	3,0049				
0,2	k	377	412	365	587	508	587	449,8	365	3,0049
	f	3,0049	3,0049	3,0049	3,0049	3,0049				
0,3	k	317	346	569	312	460	569	400,8	312	2,8055
	f	3,0049	3,0049	2,0079	3,0049	3,0049				
0,4	k	267	268	271	159	477	477	288,4	159	3,00492
	f	3,0049	3,0049	3,0049	3,0049	3,005				
0,5	k	306	211	352	394	175	394	287,6	175	3,00492
	f	3,0049	3,0049	3,005	3,0049	3,0049				
0,6	k	284	320	217	260	247	320	265,6	217	3,0049
	f	3,0049	3,0049	3,0049	3,0049	3,0049				
0,7	k	231	196	199	137	367	367	226	137	3,00488
	f	3,0049	3,0049	3,0049	3,0048	3,0049				
0,8	k	170	319	322	303	116	322	246	116	3,00486
	f	3,0049	3,0049	3,0048	3,0049	3,0048				
0,9	k	162	289	241	376	139	376	241,4	139	3,0049
	f	3,0049	3,0049	3,0049	3,0049	3,0049				
1	k	326	173	242	203	180	326	224,8	173	2,80552
	f	3,005	3,005	2,0078	3,0049	3,0049				

Dengan k = konvergen pada iterasi ke- , dan f = nilai *fitness* saat konvergen

Dari tabel hasil pengujian tersebut, dapat dibuat grafik untuk lebih mempermudah pemahaman hasil pengujian yang dapat dilihat pada Gambar 6.3 untuk hasil *fitness* dan Gambar 6.4 untuk hasil konvergensi



Gambar 6.3 Grafik *fitness* hasil pengujian probabilitas *crossover*



Gambar 6.4 Grafik konvergensi hasil pengujian probabilitas *crossover*

Dari grafik pertama, terlihat nilai *fitness* terbaik didapatkan pada rentang probabilitas *crossover* 0,4 hingga 0,9. Sedangkan pada grafik kedua terlihat jika konvergensi cenderung lebih cepat terjadi pada nilai probabilitas *crossover* yang besar dan stabil mulai pada probabilitas 0,7.

Hasil pengujian tersebut dapat dikarenakan semakin banyaknya *crossover* yang terjadi pada satu generasi mengakibatkan banyaknya antibodi baru yang terbentuk sehingga jumlah populasi meningkat. Akibatnya jika jumlah populasi terlalu banyak telah ditunjukkan pada pengujian jumlah populasi awal. Sedangkan jika proses *crossover* yang terjadi terlalu jarang dapat meningkatkan jumlah iterasi yang diperlukan untuk mencapai konvergensi, hal ini dikarenakan kemungkinan untuk mendapatkan antibodi yang lebih baik dari hasil *crossover* akan lebih kecil jika nilai probabilitas *crossover* juga kecil.

Dari hasil pengujian nilai probabilitas *crossover*, dapat diambil parameter terbaik ialah pada nilai 0,7 karena tingginya nilai *fitness* dan konvergensi yang relatif cepat.

6.3 Pengujian probabilitas mutasi

Pengujian probabilitas mutasi bertujuan untuk mendapatkan nilai probabilitas mutasi yang paling optimal. Pengujian dilakukan dengan menguji algoritma pada probabilitas mutasi dari 0,0 hingga 1,0 dengan kelipatan 0,1. Tiap-tiap nilai probabilitas akan diuji sebanyak lima kali dan hasil *fitness* dan kecepatan konvergensi akan dicatat untuk mendapatkan nilai probabilitas mutasi yang paling optimal. Parameter yang digunakan untuk pengujian nilai probabilitas mutasi ialah sebagai berikut,

1. Banyak iterasi = 1000
2. Jumlah populasi awal = 70
3. probabilitas *crossover* = 0,7

4. probabilitas mutasi = 0,0 s.d. 1,0 dengan pengujian setiap kelipatan 0,1
5. konstanta *affinity* = 0,3
6. konstanta *similarity* = 0,9
7. konstanta *death factor* = 0,5

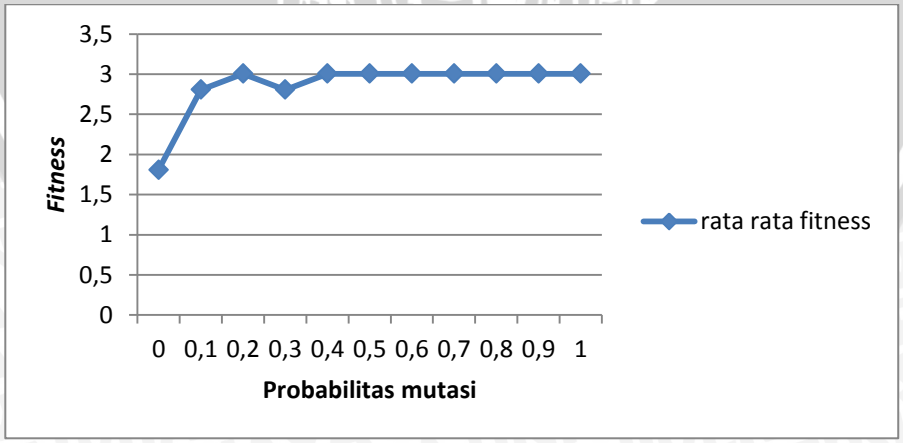
Hasil pengujian ditampilkan pada Tabel 6.3

Tabel 6.3 Hasil pengujian probabilitas mutasi

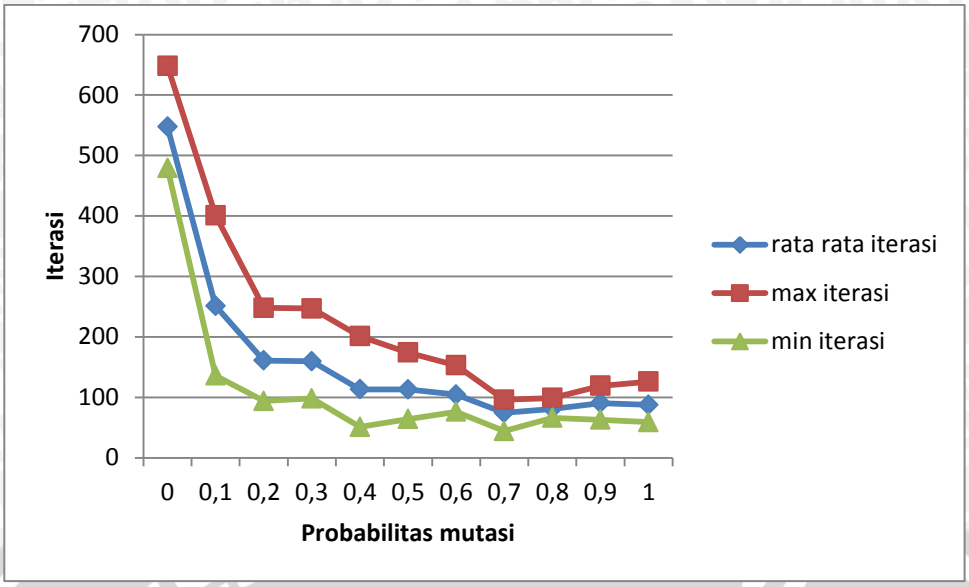
Probabilitas mutasi		Uji coba ke-					Max iterasi	Rata iterasi	Min iterasi	Rata *fitness*
		1	2	3	4	5				
0	k	479	648	528	581	501	648	547,4	479	1,8058
	f	2,005	2,0051	1,007	2,0068	2,0051				
0,1	k	401	177	314	227	136	401	251	136	2,80562
	f	2,0086	3,0048	3,0049	3,0049	3,0049				
0,2	k	248	110	184	168	94	248	160,8	94	3,0049
	f	3,0048	3,0049	3,0050	3,0049	3,0049				
0,3	k	247	229	125	99	98	247	159,6	98	2,80566
	f	3,0050	2,0088	3,0049	3,0048	3,0048				
0,4	k	125	76	51	201	113	201	113,2	51	3,0049
	f	3,0049	3,0048	3,0049	3,005	3,0049				
0,5	k	174	99	122	106	64	174	113	64	3,00494
	f	3,005	3,0049	3,005	3,0049	3,0049				
0,6	k	85	76	153	79	129	153	104,4	76	3,00494
	f	3,0049	3,0049	3,005	3,0049	3,005				
0,7	k	90	96	64	44	78	96	74,4	44	3,00488
	f	3,0049	3,0049	3,0048	3,0049	3,0049				
0,8	k	66	69	95	99	76	99	81	66	3,0049
	f	3,0049	3,0049	3,0049	3,0049	3,0049				
0,9	k	119	76	101	63	93	119	90,4	63	3,00494
	f	3,0049	3,0049	3,005	3,005	3,0049				
1	k	69	126	74	110	59	126	87,6	59	3,00496
	f	3,0049	3,005	3,005	3,005	3,0049				

Dengan k = konvergen pada iterasi ke- , dan f = nilai *fitness* saat konvergen

Hasil pengujian dapat digambarkan pada Gambar 6.5 untuk *fitness* dan Gambar 6.6 untuk konvergensi untuk lebih mempermudah pemahaman.



Gambar 6.5 Grafik *fitness* hasil pengujian probabilitas mutasi



Gambar 6.6 Grafik konvergensi hasil pengujian probabilitas mutasi

Dari grafik pertama, terlihat *fitness* cukup stabil setelah probabilitas 0,4. Dari grafik kedua dapat dilihat secara umum kecepatan konvergensi cenderung lebih cepat seiring dengan besarnya nilai probabilitas mutasi.

Hasil yang didapat dari pengujian ini dapat disebabkan karena semakin banyak terjadi mutasi, maka kemungkinan untuk mendapatkan antibodi yang lebih baik dari mutasi akan semakin besar, selain itu juga dapat mencegah hilangnya gen atau mengembalikan gen yang hilang dari populasi, sehingga memperbesar kemungkinan untuk mendapatkan antibodi yang lebih baik.

Hilangnya gen dari generasi dapat memperburuk hasil *fitness* algoritma. Dilihat dari hasil pengujian probabilitas mutasi 0,0 atau tidak adanya proses mutasi sama sekali sehingga tidak ada yang dapat mengembalikan gen yang hilang dari populasi. Hal tersebut mengakibatkan kombinasi gen pada kromosom menjadi sangat terbatas, menyebabkan terjadinya konvergensi prematur karena tidak dapat memperoleh gen yang optimal.

Dari pengujian probabilitas mutasi ini, didapatkan nilai yang paling optimal berada pada 0,7. Meskipun hasil *fitness* sudah cukup stabil pada nilai lebih dari 0,4, pada nilai 0,7 mendapatkan hasil konvergensi yang relatif cepat. Meskipun dapat memilih nilai yang paling besar untuk lebih memberi peluang konvergensi yang lebih cepat, tetapi semakin besar nilai probabilitas mutasi berarti semakin banyak antibodi yang akan mengalami proses mutasi sehingga menambah waktu eksekusi program.

6.4 Pengujian konstanta *affinity*

Pengujian konstanta *affinity* bertujuan untuk mendapatkan nilai konstanta *affinity* yang paling optimal. Nilai konstanta *affinity* yang diuji dimulai dari nilai

0,0 hingga 1,0 dengan pengujian pada setiap kelipatan 0,1. Parameter yang digunakan dalam pengujian konstanta *affinity* ini ialah sebagai berikut,

1. Banyak iterasi = 1000
2. Jumlah populasi awal = 50
3. probabilitas *crossover* = 0,7
4. probabilitas mutasi = 0,7
5. konstanta *affinity* = 0,0 s.d. 1,0 dengan pengujian setiap kelipatan 0,1
6. konstanta *similarity* = 0,9
7. konstanta *death factor* = 0,5

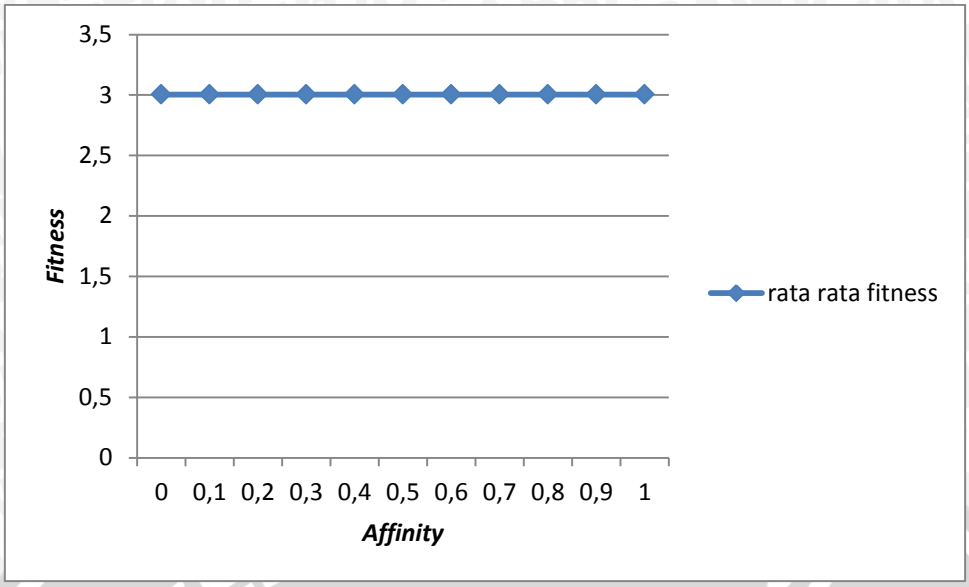
Hasil pengujian konstanta *affinity* ditampilkan pada Tabel 6.4

Tabel 6.4 Hasil pengujian konstanta *affinity*

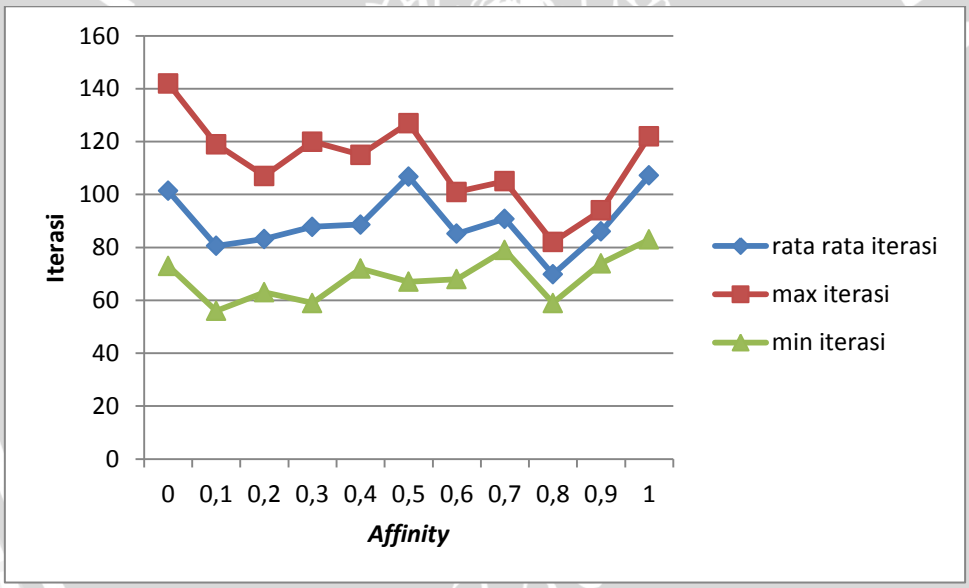
affinity		Uji coba ke- 1	2	3	4	5	Max iterasi	Rata iterasi	Min iterasi	Rata *fitness*
	k	73	85	88	119	142				
0	f	3,0049	3,0049	3,0049	3,0049	3,0049	142	101,4	73	3,0049
	k	119	56	96	58	74				
0,1	f	3,0050	3,0049	3,0049	3,0049	3,0049	119	80,6	56	3,00492
	k	63	71	107	86	89				
0,2	f	3,0050	3,0049	3,0049	3,0049	3,0049	107	83,2	63	3,00492
	k	59	109	86	65	120				
0,3	f	3,0049	3,0049	3,0050	3,0049	3,0049	120	87,8	59	3,00492
	k	72	105	115	78	73				
0,4	f	3,005	3,0049	3,005	3,005	3,0049	115	88,6	72	3,00496
	k	116	67	100	124	127				
0,5	f	3,005	3,0049	3,0049	3,0049	3,005	127	106,8	67	3,00494
	k	68	101	86	72	99				
0,6	f	3,0049	3,005	3,0049	3,0049	3,005	101	85,2	68	3,00494
	k	84	105	89	79	97				
0,7	f	3,0048	3,0049	3,005	3,0049	3,005	105	90,8	79	3,00492
	k	59	82	66	62	80				
0,8	f	3,0049	3,0049	3,0049	3,0049	3,0049	82	69,8	59	3,0049
	k	92	94	83	74	87				
0,9	f	3,0049	3,0049	3,0049	3,0049	3,005	94	86	74	3,00492
	k	103	114	114	122	83				
1	f	3,0049	3,0049	3,005	3,005	3,0049	122	107,2	83	3,00494

Dengan k = konvergen pada iterasi ke- , dan f = nilai *fitness* saat konvergen

Untuk lebih mempermudah pemahaman hasil pengujian, dapat ditampilkan dengan grafik pada Gambar 6.7 dan Gambar 6.8.



Gambar 6.7 Grafik *fitness* hasil pengujian konstanta *affinity*



Gambar 6.8 Grafik konvergensi hasil pengujian konstanta *affinity*

Dari grafik pertama dapat dilihat jika nilai konstanta *affinity* tidak terlalu berpengaruh pada hasil *fitness* karena grafik menunjukkan nilai *fitness* yang stabil berkisar di angka 3,0. Konstanta *affinity* berpengaruh pada kecepatan konvergensi. Dari grafik kedua dapat terlihat kecepatan konvergensi yang bervariasi pada setiap nilai konstanta. Nilai terbaik untuk konvergensi didapatkan pada nilai 0,8,

Hasil pengujian dapat dikarenakan konstanta *affinity* merupakan parameter untuk mengatur kecondongan algoritma seleksi. Nilai yang kecil menunjukkan kecondongannya pada faktor kedekatan antar antibodi, sedangkan nilai yang besar menunjukkan kecondongannya pada nilai *fitness*. Nilai menengah

menunjukkan kecondongan yang seimbang antara faktor kedekatan antar antibodi dan nilai *fitness* sebagai faktor seleksi.

Dari hasil pengujian, didapatkan nilai yang paling optimal berada pada rentang nilai 0,8. Pada nilai 0,8, kecondongan algoritma ialah 80 % condong pada nilai *fitness* dan 20 % sisanya ialah pada konsentrasi populasi. Hal ini menunjukkan bahwa faktor konsentrasi populasi juga dapat membantu kecepatan konvergensi.

6.5 Pengujian konstanta *similarity*

Pengujian konstanta *similarity* bertujuan untuk mendapatkan nilai konstanta *similarity* yang paling optimal. Nilai konstanta *similarity* yang diujicoba mulai dari 0,1 hingga 1,0 dengan kelipatan 0,1 tiap pengujian. Parameter yang digunakan dalam pengujian ini ialah sebagai berikut,

1. Banyak iterasi = 1000
2. Jumlah populasi awal = 70
3. probabilitas *crossover* = 0,7
4. probabilitas mutasi = 0,7
5. konstanta *affinity* = 0,8
6. konstanta *similarity* = 0,1 hingga 1 dengan pengujian tiap kelipatan 0,1
7. konstanta *death factor* = 0,5

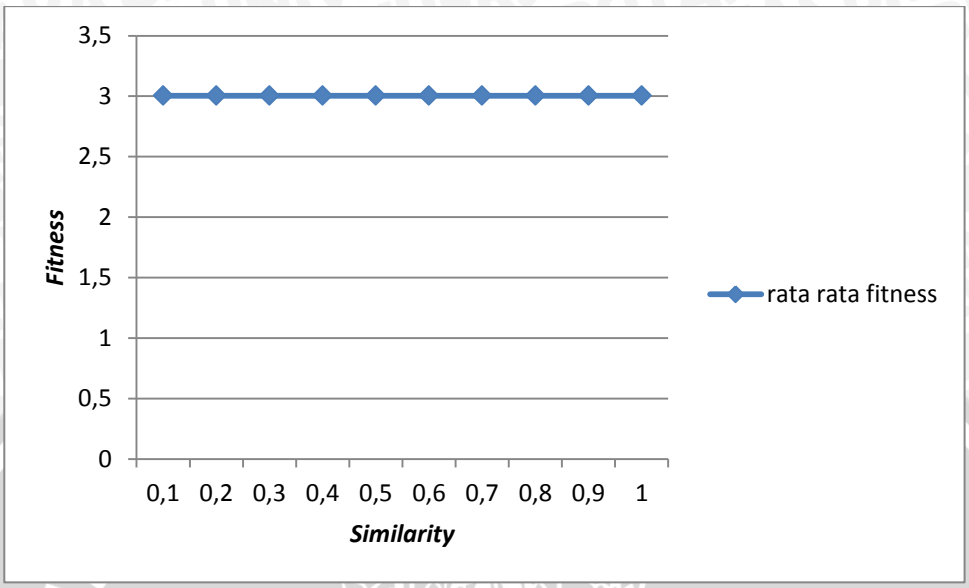
Hasil pengujian ditampilkan pada Tabel 6.5.

Tabel 6.5 Hasil pengujian konstanta *similarity*

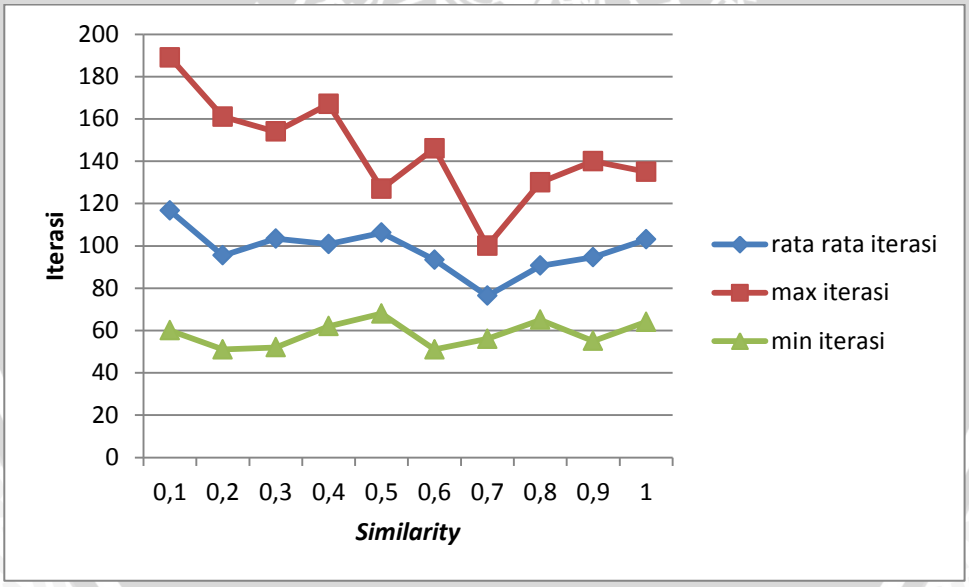
similarity		Uji coba ke-					Max iterasi	Rata iterasi	Min iterasi	Rata *fitness*
		1	2	3	4	5				
0,1	k	60	173	67	94	189	189	116,6	60	3,00494
	f	3,0049	3,0050	3,0049	3,0049	3,0050				
0,2	k	109	161	84	51	72	161	95,4	51	3,00492
	f	3,0050	3,0049	3,0049	3,0049	3,0049				
0,3	k	77	154	52	122	112	154	103,4	52	3,00494
	f	3,0049	3,0050	3,0049	3,0049	3,0050				
0,4	k	102	80	167	93	62	167	100,8	62	3,00492
	f	3,0049	3,0049	3,005	3,0049	3,0049				
0,5	k	123	68	118	95	127	127	106,2	68	3,00494
	f	3,005	3,0049	3,0049	3,0049	3,005				
0,6	k	126	69	75	146	51	146	93,4	51	3,00488
	f	3,005	3,0049	3,0049	3,0047	3,0049				
0,7	k	70	100	96	60	56	100	76,4	56	3,00492
	f	3,0049	3,0049	3,005	3,0049	3,0049				
0,8	k	65	71	130	117	70	130	90,6	65	3,00488
	f	3,0048	3,0049	3,0049	3,0049	3,0049				
0,9	k	140	67	55	83	128	140	94,6	55	3,00488
	f	3,0049	3,0049	3,0048	3,0049	3,0049				
1	k	133	109	74	135	64	135	103	64	3,00496
	f	3,005	3,005	3,0049	3,005	3,0049				

Dengan k = konvergen pada iterasi ke- , dan f = nilai *fitness* saat konvergen

Data hasil pengujian tersebut dapat ditampilkan dalam bentuk grafik, seperti yang ditunjukkan pada Gambar 6.9 untuk pengujian *fitness* dan Gambar 6.10 untuk pengujian *similarity*



Gambar 6.9 Grafik pengujian *fitness* konstanta *similarity*



Gambar 6.10 Grafik pengujian konvergensi konstanta *similarity*

Dari grafik pertama, terlihat nilai *similarity* tidak terlalu berpengaruh pada nilai *fitness*, terlihat pada grafik nilai *fitness* yang pada semua nilai berada pada angka 3. Hal ini berbeda dengan keadaan pada grafik kedua, yang menunjukkan variasi konvergensi pada tiap nilai konstanta *similarity*. Terlihat di grafik kedua, nilai terbaik berada pada konstanta 0,7.

Hasil dari pengujian ini dapat disebabkan karena konstanta *similarity* merupakan batas dimana antibodi satu dengan lainnya dianggap dekat sehingga

dapat mempengaruhi nilai konsentrasi untuk proses seleksi imun. Nilai yang kecil menunjukkan meskipun antibodi satu dengan lainnya memiliki cukup banyak perbedaan, kedua antibodi tersebut dapat dianggap dekat, sedangkan nilai konstanta yang tinggi membuat dua antibodi harus memiliki kemiripan yang tinggi agar dianggap dekat.

Dari hasil pengujian memperlihatkan jika hasil yang paling baik jika nilai konstanta *similarity* tidak terlalu besar, juga tidak terlalu kecil. Pada pengujian ini terlihat konstanta pada nilai 0,7 mendapatkan hasil *fitness* yang baik serta memiliki rentang maksimum-minimum konvergensi yang relatif baik.

6.6 Pengujian konstanta *death factor*

Pengujian konstanta *death factor* bertujuan untuk mendapatkan nilai yang paling optimal. Nilai konstanta death facot yang diujicoba mulai dari 0,1 hingga 1 dengan pengujian setiap kelipatan 0,1. Parameter yang digunakan dalam pengujian ini ialah sebagai berikut,

1. Banyak iterasi = 1000
2. Jumlah populasi awal = 70
3. probabilitas *crossover* = 0,7
4. probabilitas mutasi = 0,7
5. konstanta *affinity* = 0,8
6. konstanta *similarity* = 0,7
7. konstanta *death factor* = 0,1 s.d. 1 dengan pengujian tiap kelipatan 0,1

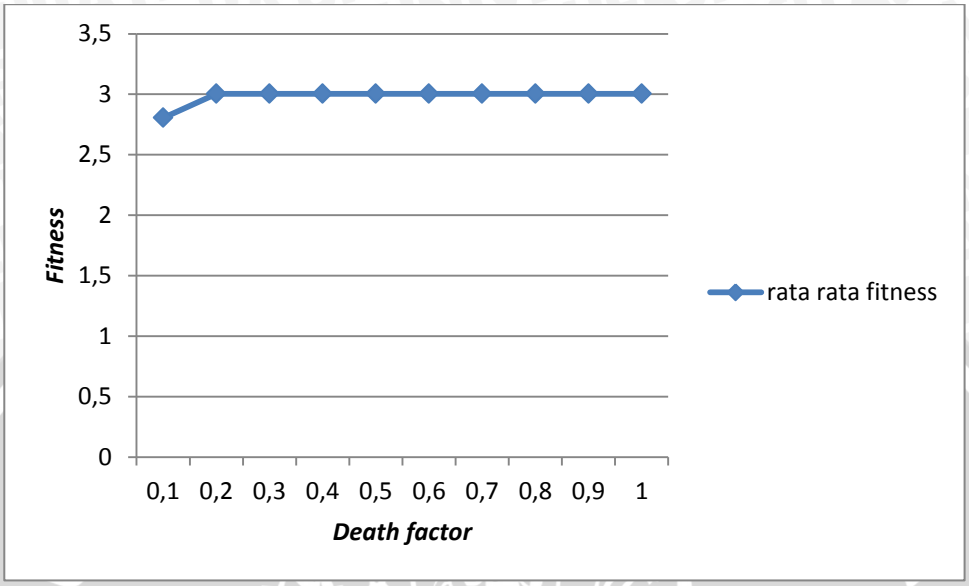
Hasil pengujian ditampilkan pada Tabel 6.6

Tabel 6.6 Hasil pengujian konstanta *death factor*

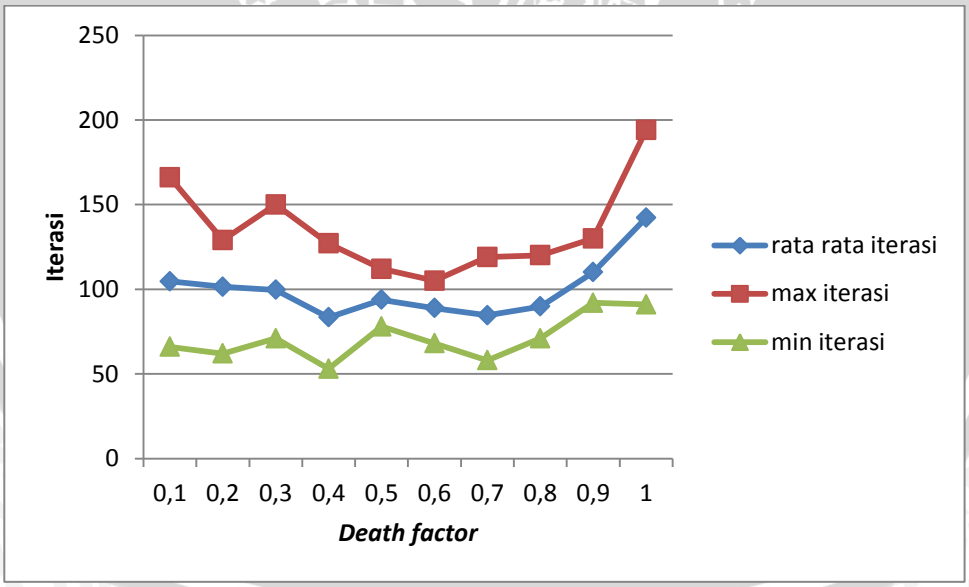
death factor		Uji coba ke-					Max iterasi	Rata iterasi	Min iterasi	Rata *fitness*
		1	2	3	4	5				
	k	127	66	95	69	166				
0,1	f	3,00490	3,00490	3,00490	3,00490	2,00890	166	104,6	66	2,8057
	k	73	127	129	116	62				
0,2	f	3,0049	3,0049	3,0049	3,0049	3,0040	129	101,4	62	3,00472
	k	86	118	73	150	71				
0,3	f	3,0049	3,0049	3,0049	3,0050	3,0049	150	99,6	71	3,00492
	k	102	69	127	66	53				
0,4	f	3,0049	3,0049	3,005	3,0049	3,0049	127	83,4	53	3,00492
	k	112	83	78	104	92				
0,5	f	3,0049	3,0049	3,005	3,0049	3,0049	112	93,8	78	3,00492
	k	96	105	86	89	68				
0,6	f	3,0049	3,0049	3,0049	3,0049	3,0049	105	88,8	68	3,0049
	k	58	77	119	97	72				
0,7	f	3,0049	3,0049	3,0049	3,0049	3,0049	119	84,6	58	3,0049
	k	92	87	120	79	71				
0,8	f	3,0049	3,005	3,005	3,0049	3,0049	120	89,8	71	3,00494
	k	92	130	116	116	96				
0,9	f	3,0049	3,005	3,005	3,0049	3,0049	130	110	92	3,00494
	k	95	91	194	143	188				
1	f	3,005	3,0049	3,0049	3,005	3,005	194	142,2	91	3,00496

Dengan k = konvergen pada iterasi ke- , dan f = nilai *fitness* saat konvergen

Dari hasil pengujian tersebut, dapat ditampilkan dalam bentuk grafik untuk mempermudah pemahaman yang ditunjukkan pada Gambar 6.11 untuk grafik *fitness*, dan Gambar 6.12 untuk grafik konvergensi.



Gambar 6.11 Grafik pengujian *fitness* konstanta *death factor*



Gambar 6.12 Grafik pengujian konvergensi konstanta *death factor*

Dari grafik pertama dapat dilihat jika selain nilai yang terlalu kecil, *death factor* tidak terlalu berpengaruh pada nilai *fitness*. Dari grafik kedua terlihat sedikit variasi konvergensi pada tiap nilai konstanta *death factor*, tetapi menunjukkan kestabilan kecepatan konvergensi, terutama pada nilai 0,5 hingga 0,8.

Hasil pengujian tersebut dapat disebabkan karena konstanta *death factor* merupakan konstanta yang mengatur seberapa banyak antibodi akan dihapus dari populasi pada saat seleksi imun. Pada pengujian ini diberi batasan pada jumlah populasi, yaitu populasi maksimum sebanyak 100. Jika tidak diberi batasan populasi maksimum, jumlah populasi dapat bertambah hingga sangat banyak sampai pada titik program tidak dapat berjalan dengan baik (*crash*), karena itu disarankan untuk memberi batasan populasi maksimum untuk mencegah hal tersebut terjadi. Nilai *death factor* yang terlalu kecil dapat menyebabkan jumlah populasi yang menggelembung dengan cepat, sedangkan nilai yang terlalu besar mengakibatkan terlalu banyak antibodi yang akan mati setiap iterasi.

Meskipun grafik konvergensi terlihat cukup stabil, dipilih nilai 0,7 sebagai nilai paling optimal sebagai konstanta *death factor*.

6.7 Analisa global

Dari hasil pengujian, didapatkan parameter paling optimal yaitu sebagai berikut,

1. Banyak iterasi = 1000
2. Jumlah populasi awal = 70
3. probabilitas *crossover* = 0,7
4. probabilitas mutasi = 0,7
5. konstanta *affinity* = 0,8
6. konstanta *similarity* = 0,7
7. konstanta *death factor* = 0,7

Dengan parameter tersebut, didapatkan contoh hasil menu untuk 2 hari seperti yang ditampilkan pada Tabel 6.7

Tabel 6.7 Contoh hasil menu makanan dari solusi terbaik

Hari 1			
Nasi putih	tumis kangkung	telur goreng	teh
Nasi Goreng	mie telur	usus ayam goreng	air putih
Nasi Kuning	sayur sop	tahu goreng	air putih
Biaya : Rp. 288.800			
Protein : 1945,60 | kecukupan : 1,01			
Lemak : 2119,45 | kecukupan : 0,91			
Karbohidrat : 8823,41 | kecukupan : 0,93			
Hari 2			
Nasi Pecel	sayur asem	telur dadar	teh
Nasi putih	mie goreng sawi	pindang goreng	teh
Nasi Pecel	sayur bayam	dadar jagung	susu
Biaya : Rp. 302.100			
Protein : 2033,38 | kecukupan : 1,06			

Lemak : 2497,74 | kecukupan : 1,08
Karbohidrat : 8570,60 | kecukupan : 0,90

Terlihat meskipun nilai gizinya telah mencukupi kebutuhan anak sesuai dengan AKG, tetapi biaya yang dihasilkan masih cukup tinggi. Jika dibandingkan dengan Data menu makanan dari panti asuhan pada Tabel 6.8 dapat dilihat bahwa hasil penelitian ini dapat memberi nilai kecukupan gizi yang lebih baik, hanya saja memilki kelemahan pada tingginya biaya yang harus dikeluarkan.

Selisih biaya yang tinggi dapat disebabkan karena dimasukkannya variabel variasi menu makanan perhitungan nilai *fitness*, serta tidak adanya pembatas khusus untuk tiap jenis makanan (misal, jika makanan pokok adalah nasi pecel, maka tidak perlu ada sayuran), sehingga algoritma akan menghindari kombinasi menu yang berbiaya rendah tetapi tidak ada variasi menu didalamnya.

Tabel 6.8 Perkiraan gizi & biaya dari menu panti asuhan

Hari 1			
Nasi Pecel	-	Tahu goreng	Teh
Nasi	Tumis kacang panjang	Tahu goreng	Sirup
Nasi	Mie Goreng Sawi	Pindang goreng	Susu
Biaya : Rp. 209.000			
Protein : 2633,4 | kecukupan : 1,37			
Lemak : 1710,38 | kecukupan : 0,73			
Karbohidrat : 8230,87 | kecukupan : 0,86			
Hari 2			
Nasi	Tumis kangkung	Tempe goreng	Teh
Nasi	Sayur sop	Telur dadar	Air putih
Nasi	Rawon	Tempe goreng	Air putih
Biaya Rp. 214.700			
Protein : 1187,88 | kecukupan : 0,61			
Lemak : 1174,96 | kecukupan : 0,50			
karbohidrat : 6230,86 | kecukupan : 0,65			

Dari Tabel 6.8 terlihat menu dari panti asuhan memiliki kecukupan gizi yang kurang baik. Terlihat masih ada kelebihan gizi, dan banyak yang kekurangan kandungan gizi. Dengan hasil penelitian ini diharapkan dapat memperbaiki kandungan gizi dari tiap menu makanan yang dibuat untuk panti asuhan.

BAB 7 PENUTUP

Bab ini membahas tentang kesimpulan yang diperoleh dari pembuatan dan pengujian sistem Optimasi Gizi pada Anak Panti Asuhan dengan *Immune Genetic Algorithm* serta saran yang dapat digunakan untuk pengembangan penelitian tersebut selanjutnya.

7.1 Kesimpulan

Berdasarkan hasil pengujian dan analisis hasil sistem Optimasi Gizi pada Anak Panti Asuhan dengan *Immune Genetic Algorithm,*dapat diambil kesimpulan sebagai berikut,

1. *Immune Genetic Algorithm* dapat digunakan untuk mengoptimasi gizi pada anak panti asuhan dengan cara merepresentasikan solusi yang berupa menu makanan dan biayanya sebagai kromosom yang akan diproses oleh algoritma. Satu kromosom akan merepresentasikan menu santapan dalam bentuk deretan nomor indeks makanan sehingga membentuk satu menu makanan untuk beberapa hari sekaligus, lengkap untuk makan pagi, siang, maupun malam dengan masing-masing mengandung makanan berupa makanan pokok, sayur, lauk, serta minuman. Kromosom terbaik dilihat dari pemenuhan gizi, kombinasi menu, dan biaya yang paling minimal, akan dijadikan sebagai rekomendasi menu makanan untuk optimasi gizi pada anak panti asuhan.
2. Hasil dari pengujian menunjukkan bahwa semakin banyak jumlah populasi, akan semakin cepat terjadi konvergensi, tetapi nilai *fitness* terbaik didapatkan pada jumlah populasi awal 70. Pada pengujian probabilitas *crossover* juga menunjukkan semakin tinggi probabilitas, maka akan cenderung lebih cepat terjadi konvergensi, hanya saja jika nilainya terlalu besar akan berakibat pada semakin banyaknya terjadi konvergensi prematur. Nilai paling optimal untuk probabilitas *crossover* berada pada nilai 0,7. Dari pengujian probabilitas mutasi, terlihat kecepatan konvergensi dan nilai *fitness* cenderung membaik seiring semakin besarnya nilai probabilitas. Nilai paling optimal untuk probabilitas mutasi ialah 0,7. Konstanta *affinity* tidak terlalu berpengaruh pada nilai *fitness*, melainkan lebih berpengaruh pada kecepatan konvergensi. Konvergensi terbaik terjadi pada nilai konstanta *affinity* 0,8. Untuk konstanta *similarity* menunjukkan nilai *fitness* yang stabil pada nilai 0,7. Sedangkan untuk konstanta *death factor*, hasil pengujian menunjukkan bahwa *death factor* tidak terlalu berpengaruh pada hasil nilai *fitness*. Konvergensi juga terlihat cukup stabil, nilai paling optimal berada pada 0,7. Dengan nilai parameter tersebut, hasil dari algoritma menunjukkan jika menu yang dihasilkan sudah dapat memberi kecukupan gizi yang baik, hanya saja biaya yang dihasilkan masih cukup tinggi.

7.2 Saran

Saran yang dapat digunakan untuk pengembangan penelitian ini selanjutnya ialah sebagai berikut,

1. Pada perhitungan nilai *fitness*, dapat dilakukan perbaikan untuk lebih menyeimbangkan antara nilai gizi dengan harga atau biaya, sehingga dapat tetap memberikan rekomendasi menu yang dapat memenuhi gizi dengan biaya yang lebih minimal.
2. Pada proses seleksi, dapat dilakukan perbaikan agar perubahan jumlah populasi tidak terlalu jauh dari jumlah populasi awal, sehingga meminimalkan kemungkinan program tidak berjalan dengan baik.
3. Penelitian ini lebih mengutamakan kecukupan gizi harian untuk satu populasi, diharapkan untuk penelitian selanjutnya lebih memperhatikan kebutuhan gizi tiap individu tiap kali makan sehingga menu yang dihasilkan dapat lebih optimal untuk kebutuhan harian, maupun kebutuhan tiap kali makan baik untuk keseluruhan populasi maupun tiap individu.
4. Data menu makanan yang digunakan sebaiknya lebih lengkap dan bervariasi agar sistem memiliki lebih banyak pilihan, sehingga diharapkan hasil keluaran menu lebih optimal.
5. Lebih memperhatikan kombinasi menu makanan agar dapat menghasilkan kombinasi menu makanan yang lebih realistis dengan tetap memberi kecukupan gizi yang baik serta biaya yang minimum.

DAFTAR PUSTAKA

Bondal, A.A., 2008. *Artificial Immune Systems Applied to Job Shop Scheduling*. Russ College of Engineering and Technology of Ohio University.

Brownlee, J., 2011. *Clever Algorithms: Nature-Inspired Programming Recipes*. 1st ed. LuLu.

Fang, X. et al., 2012. An Artificial Immune System for Phishing Detection. In *Evolutionary Computation (CEC), 2012 IEEE Congress on*. Brisbane, 2012. IEEE.

Fauzi, C.A., 2012. Analisis Pengetahuan dan Perilaku Gizi Seimbang Menurut Pesan ke-6, 10, 11, 12 dari Pedoman Umum Gizi Seimbang (PUGS) pada Remaja. *Jurnal Kesehatan Reproduksi*, 3(2), pp.91 - 105.

Indonesia, 2011. *Peraturan Mentri Sosial No. 3 Tentang Standar Nasional Pengasuhan Anak Untuk Lembaga Kesejahteraan Sosial Anak*. Jakarta: Kementrian Sosial.

Indonesia, 2013. *Peraturan Mentri Kesehatan Republik Indonesia Nomor 75 Tahun 2013 Tentang Angka Kecukupan Gizi yang Dianjurkan bagi Bangsa Indonesia*. Jakarta: Kementrian Kesehatan Republik Indonesia.

Jin, Z. & Fan, H., 2013. A New Immune Genetic Algorithm for 0-1 Knapsack Problem. *Sixth International Symposium on Computational Inteligence and Design*, pp.31-33.

Juan, S., Dan, S.Y. & Min, X., 2009. Research on Application of IGA (Immune Genetic Algorithm) to the Solution of Course-Timetabling Problem. In *International Conference on Computer Science and Education*., 2009. IEEE.

Laksmiarti, T. et al., 2013. *Riset Kesehatan Dasar Dalam Angka Riskesdas 2013*. Jawa Timur: Badan Penelitian dan Pengembangan Kesehatan Kementrian Kesehatan RI.

Lei, W. & Licheng, J., 2000. Immune Evolutionary Algorithms. In *Signal Processing Proceedings, 2000. WCCC-ICSP 2000. 5th International Conference on (Volume:3)*. Beijing, 2000. IEEE.

Mahmudy, W.F., 2013. *Algoritma Genetika*.

Mahmudy, W.F., 2014. Optimasi Penjadwalan Two-Stage Assembly Flowshop Menggunakan Algoritma Genetika yang Dimodifikasi. In *Konferensi Nasional Sistem Informasi (KNSI)*. Makassar, 2014.

Muchtadi, D., 2014. *Pengantar Ilmu Gizi*. Bandung: Alfabeta.

Pratiwi, M.I., Mahmudy, W.F. & Dewi, C., 2014. Implementasi Algoritma Genetika pada Optimasi Biaya Pemenuhan Kebutuhan Gizi. *Repository Jurnal Mahasiswa PTIIK*, 4(6).

Sari, A.P., Mahmudy, W.F. & Dewi, C., 2014. Optimasi Asupan Gizi pada Ibu Hamil dengan Menggunakan Algoritma Genetika. *Repositori Jurnal Mahasiswa PTIIK UB*, 4(5).

Suci, W.W., Mahmudy, W.F. & Putri, R.R.M., 2015. Optimasi Biaya Pemenuhan Gizi dan Nutrisi pada Manusia Lanjut Usia menggunakan Algoritma Genetika. *Repositori Jurnal Mahasiswa PTIIK UB*, 5(17).

Sutojo, T., Mulyanto, E. & Suhartono, V., 2011. *Kecerdasan Buatan*. Yogyakarta: Andi.

Wang, L., Jiao, L. & Member, S., 1998. The Immune Genetic Algorithm and Its Convergence. In *Signal Processing Proceedings, 1998. ICSP '98. 1998 Fourth International Conference on (Volume:2)*. Beijing, 1998. IEEE.

Weckman, G., Bondal, A.A., Rinder, M.M. & Young II, W.A., 2012. Applying a hybrid artificial immune systems to the job shop scheduling problem. *Neural Comput & Applic*, (21), pp.1465 - 1475.

William, A., 2010. *Gambaran Status Gizi Anak di Panti Asuhan Yayasan Terima Kasih Abadi Kecamatan Medan Barat Tahun 2010*. Medan: Fakultas Kedokteran Universitas Sumatera Utara.

Yan-cong, Z. et al., 2011. Research on Test Paper Auto-Generating Based on Immune Genetic Algorithm. In *Control and Decision Conference (CCDC), 2011 Chinese*. Mianyang, 2011. IEEE.

Yang, Y., 2011. Application of Artificial Immune System in Handwritten Rassian Uppercase Character Recognition. In *Computer Science and Service System (CSSS), 2011 International Conference on*. Nanjing, 2011. IEEE.

Zukhri, Z., 2014. *Algoritma Genetika Metode Komputasi Evolusioner untuk Menyelesaikan Masalah Optimas*. 1st ed. Yogyakarta: Andi.

LAMPIRAN A PERHITUNGAN MANUAL

A.1 Angka Kecukupan Gizi (AKG)

Angka Kecukupan Gizi (AKG) merupakan suatu kecukupan rata rata zat gizi setiap hari bagi semua orang menurut golongan umur, jenis kelamin, ukuran tubuh, serta aktifitas tubuh untuk mencapai derajat kesehatan yang optimal.

Kelompok Umur	BB* (kg)	TB* (cm)	Energi (kkal)	Protein (g)	Lemak (g)	Karbohidrat (g)
Bayi/Anak						
0-6 bulan	6	61	550	12	34	58
7-11 bulan	9	71	725	18	36	82
1-3 tahun	13	91	1125	26	44	155
4-6 tahun	19	112	1600	35	62	220
7-9 tahun	27	130	1850	49	72	254
Laki-laki						
10-12 tahun	34	142	2100	56	70	289
13-15 tahun	46	158	2475	72	83	340
16-18 tahun	56	165	2675	66	89	368
19-29 tahun	60	168	2725	62	91	375
30-49 tahun	62	168	2625	65	73	394
50-64 tahun	62	168	2325	65	65	349
65-80 tahun	60	168	1900	62	53	309
80+ tahun	58	168	1525	60	42	248
Perempuan						
10-12 tahun	36	145	2000	60	67	275
13-15 tahun	46	155	2125	69	71	292
16-18 tahun	50	158	2125	59	71	292
19-29 tahun	54	159	2250	56	75	309
30-49 tahun	55	159	2150	57	60	323
50-64 tahun	55	159	1900	57	53	285
65-80 tahun	54	159	1550	56	43	252
80+ tahun	53	159	1425	55	40	232

A.2 Data anak panti asuhan

Berikut data anak panti asuhan Nurul Abyadh

No	Nama	Jenis Kelamin	Umur (tahun)
1	Aldi	Laki-Laki	10
2	Firman	Laki-Laki	11
3	Yohanes	Laki-Laki	12
4	Irman	Laki-Laki	13
5	Abd Kodir F	Laki-Laki	15
6	Misbakhul Ulum	Laki-Laki	15
7	Gunawan Putra	Laki-Laki	16
8	Ikbal Hobir	Laki-Laki	16
9	Tio	Laki-Laki	16
10	Achmad Budiono	Laki-Laki	17
11	Ainur Rahman	Laki-Laki	17
12	Jonathan	Laki-Laki	17
13	Sam'an	Laki-Laki	17
14	Susanto	Laki-Laki	17
15	Wardi	Laki-Laki	18
16	Jonathan Firdaus	Laki-Laki	19
17	Novi	Perempuan	9
18	Sela Indriani	Perempuan	11
19	Srilati	Perempuan	11

No	Nama	Jenis Kelamin	Umur (tahun)
20	Dinda P.P	Perempuan	12
21	Vanisa Ayu M	Perempuan	12
22	Urwatul Wasikoh	Perempuan	13
23	Miftahul Jannah	Perempuan	14
24	Nisfatul Laely	Perempuan	14
25	Raudhatul Jannah	Perempuan	14
26	Rohmadiyah	Perempuan	15
27	Selvi L.L	Perempuan	15
28	Siti Parihah	Perempuan	15
29	Indah Sosilowati	Perempuan	16
30	Insiyah	Perempuan	16
31	Khoirotun Nisa'	Perempuan	16
32	Siti Asih	Perempuan	16
33	Ari Qurrota A	Perempuan	17
34	Eni	Perempuan	17
35	Frida Sucila	Perempuan	17
36	Mina	Perempuan	17
37	Nunung	Perempuan	17
38	Malisa	Perempuan	18

Dari tabel daftar anak panti asuhan, dapat disederhanakan lagi menjadi daftar jumlah anak berdasarkan golongan umur dan jenis kelamin sehingga lebih memudahkan untuk menghitung kebutuhan gizinya sesuai dengan tabel AKG

Data Anak			Kebutuhan Populasi dalam satu hari			
Jenis Kelamin	**Range Umur**	**Jumlah**	**Energi (kkal)**	**Protein (g)**	**Lemak (g)**	**Karbohidrat (g)**
Semua	7 - 9 tahun	1	1850	49	72	254
Laki-laki	10 - 12 tahun	3	6300	168	210	867
	13 - 15 tahun	3	7425	216	249	1020
	16 - 18 tahun	9	24075	594	801	3312
	19 - 29 tahun	1	2725	62	91	375
Perempuan	10 - 12 tahun	4	8000	240	268	1100
	13 - 15 tahun	7	14875	483	497	2044
	16 - 18 tahun	10	21250	590	710	2920
	19 - 29 tahun	0	0	0	0	0
Total kebutuhan harian			**86500**	**2402**	**2898**	**11892**
Makan pagi (25 %) *			21625	600,5	724,5	2973
Makan siang (30 %) *			25950	720,6	869,4	3567,6
Makan malam (25 %) *			21625	600,5	724,5	2973
Total kebutuhan tanpa snack (80 %) *			69200	1921,6	2318,4	9513,6

Keterangan

* = Aturan per jam makan, makan pagi 25 % dari kebutuhan total harian, siang 30 % dari kebutuhan harian, serta malam 25 % dari kebutuhan total harian. Sisa 20 % merupakan asupan yang berasal dari makanan ringan yang dapat diabaikan dalam penelitian ini.

A.3 Daftar makanan

Penelitian ini menggunakan 4 jenis daftar makanan, yaitu

A.3.1 Makanan pokok

No	Nama	Energi (kkal)	Protein (gr)	Lemak (gr)	Karbohidrat (gr)	Sumber Data	Ukuran 1 porsi (gr)	Harga (Rp)
1	Nasi Putih	178	2.1	0.1	40.6	depkes	150	0 *
2	Nasi Kuning	475	7.8	15.8	80	depkes	150	1000
3	Nasi Goreng	250	9.39	9.28	31.38	fatsecret.co.id	149	1000

Keterangan

* = Pihak panti asuhan mendapatkan beras dari sumbangan, sehingga nilai harga nasi putih bisa diabaikan (Rp 0)

A.3.2 Sayur

No	Nama	Energi (kkal)	Protein (gr)	Lemak (gr)	Karbohidrat (gr)	Sumber Data	Ukuran 1 porsi (gr)	Harga (Rp)
1	Cap cay kuah	60	1.42	2.33	8.7	fatsecret.co.id	90	2000
2	Gudeg	127	1.82	4.71	22.54	fatsecret.co.id	100	1000
3	sayur lodeh	104	3.48	2.62	17.44	depkes	150	1500
4	Rawon	119	9.6	7.4	3.48	fatsecret.co.id	100	4000
5	Sayur asem	33	1.32	1.15	5.38	fatsecret.co.id	100	1000
6	Sayur Bayam	74	5.33	4.16	6.81	depkes	150	1000
7	sayur sop	90	10.7	4	4.7	depkes	150	2000
8	Tumis kangkung	98	2.56	8.71	3.99	fatsecret.co.id	100	1000

A.3.3 Lauk

No	Nama	Energi (kkal)	Protein (gr)	Lemak (gr)	Karbohidrat (gr)	Sumber Data	Ukuran 1 porsi (gr)	Harga (Rp)
1	Ayam goreng	260	21.93	14.55	10.76	fatsecret.co.id	100	4000
2	Perkedel jagung	91	1.87	5.4	9.45	fatsecret.co.id	35	100
3	telur balado	202	10.21	16.43	3.4	depkes	150	2000
4	Tahu goreng	35	2.23	2.62	1.36	fatsecret.co.id	13	300
5	Telur dadar	93	6.48	7.33	0.42	fatsecret.co.id	6	1000
6	Ikan lele	204	14.93	12.35	12.35	depkes	150	2000
7	Tempe bacem	49	3.73	3.1	2.23	fatsecret.co.id	25	500
8	Tempe goreng	34	2	2.28	1.79	fatsecret.co.id	15	300
9	mie telur goreng	146	5.38	1.69	27.08	depkes	150	1500

A.3.4 Minuman

No	Nama	Energi (kkal)	Protein (gr)	Lemak (gr)	Karbohidrat (gr)	Sumber Data	Ukuran 1 porsi (gr)	Harga (sebelum diolah)
1	Air putih	0	0	0	0			0
2	Sirup	850	0	0.13	219.3	fatsecret.co.id		300
3	Susu	122	8.03	4.88	11.49	fatsecret.co.id		2000
4	Teh Manis	55	0	0	14.36	fatsecret.co.id		300

A.4 Inisialisasi

Proses ini akan membentuk generasi awal secara acak sesuai dengan representasi kromosom sejumlah yang sudah ditentukan

Hari	1												2											
Waktu	Pagi				Siang				Malam				Pagi				Siang				Malam			
Menu	P	S	L	M	P	S	L	M	P	S	L	M	P	S	L	M	P	S	L	M	P	S	L	M
A 1	3	8	8	1	3	4	1	1	3	10	7	4	3	4	2	2	2	10	5	3	3	7	2	2
A 2	3	4	9	2	1	10	4	1	1	8	6	4	1	10	4	4	2	10	8	1	3	7	9	4
A 3	2	8	8	4	1	9	6	1	2	5	6	3	3	2	8	3	2	2	7	3	3	7	7	4
A 4	2	8	7	3	2	10	1	1	3	2	2	2	1	5	7	4	1	4	5	1	1	11	2	2
A 5	1	9	5	2	2	4	5	2	3	7	7	4	2	5	6	1	2	7	7	4	2	9	8	2

Keterangan :

P : Makanan pokok

S : Sayur

L : Lauk

M : Minuman

Nilai tiap gen digenerasi secara random berdasarkan rentang yang sesuai dengan menunya. Misal pada daftar makanan pokok hanya terdapat 3 data, sehingga nilai yang dapat masuk pada gen makanan pokok mulai dari nilai 1, 2, dan 3.

A.5 Perhitungan nilai *fitness*

Contoh perhitungan *fitness* untuk Antibodi 1

Untuk mendapatkan nilai *fitness* diperlukan nilai selisih gizi, selisih biaya, dan nilai kesamaan.

Untuk mendapatkan nilai selisih gizi dan selisih biaya digunakan cara berikut,

Hari	1												2											
Waktu	Pagi				Siang				Malam				Pagi				Siang				Malam			
Kromosom	3	8	8	1	3	4	1	1	3	10	7	4	3	4	2	2	2	10	5	3	3	7	2	2
Nama	Nasi Goreng	Sayur sop	Tempe goreng	Air putih	Nasi Goreng	mie telur	Ayam goreng	Air putih	Nasi Goreng	Tumis kangkung	tempe bacem	Teh	Nasi Goreng	mie telur	Dadar jagung	Sirup	Nasi Pecel	Tumis kangkung	Telur dadar	Susu	Nasi Goreng	Sayur bayam	Dadar jagung	Sirup
Protein	356,82	80,56	76	0	356,82	275,88	333,336	0	356,82	104,88	141,74	0	356,82	275,88	71,06	0	300,2	104,88	246,24	298,68	356,82	109,44	71,06	0
per makan[1]	513,38				966,036				603,44				703,76				950				537,32			
per hari[2]	2082,856												2191,08											
Selisih per makan[3]	87,12				245,436				0				103,26				229,4				63,18			
Selisih per hari[4]	0												269,48											
total selisih[5]	269,48																							
Lemak	352,64	73,34	86,64	0	352,64	125,78	221,16	0	352,64	357,2	117,8	0	352,64	125,78	205,2	0,76	425,6	357,2	278,54	301,34	352,64	59,28	205,2	0,76
per makan	512,62				699,58				827,64				684,38				1362,68				617,88			
per hari	2039,84												2664,94											
Selisih per makan	211,88				169,82				103,14				0				493,28				106,62			
Selisih per hari	278,56												346,54											
total selisih	625,1																							

Karbohidrat	1192,44	455,24	68,02	0	1192,44	1529,88	163,552	0	1192,44	163,78	84,74	699,2	1192,44	1529,88	359,1	1041,58	1447,42	163,78	15,96	419,14	1192,44	337,44	359,1	1041,58
per makan				1715,7				2885,872				2140,16				4123				2046,3				2930,56
per hari												6741,732												9099,86
Selisih				1257,3				681,728				832,84				1150				1521,3				0
per hari												2771,868												0
total selisih																								2771,868
Harga	95000	38000	11400	3800	95000	57000	38000	3800	95000	38000	19000	11400	95000	57000	11400	11400	95000	38000	28500	11400	95000	38000	11400	11400
per makan				148200				193800				163400				174800				172900				155800
per hari												505400												503500
total harga																								1008900
Selisih per makan[6]				98200				143800				113400				124800				122900				105800
Selisih per hari[7]												355400												353500
total selisih																								708900
fragment *fitness*[8]				0,027023562				0,018330731				1,019619993				1,018472972				0,015156954				1,034324838

Keterangan :

[1] Kandungan gizi per makan merupakan jumlah kandungan gizi dari makanan pokok, sayur, lauk dan minuman dalam satu waktu makan

[2] Kandungan gizi per hari merupakan jumlah kandungan gizi dalam tiga kali makan (pagi – siang – malam) dalam satu hari

[3] Selisih per makan merupakan selisih kandungan gizi per makan dengan kebutuhan gizi per makan sesuai dengan waktu makan masing-masing

[4] Selisih per hari merupakan selisih kandungan gizi per hari dengan kebutuhan gizi per hari

[5] Total selisih merupakan jumlah dari selisih per hari

[6] Selisih harga per makan merupakan selisih biaya tiap makan dengan biaya harapan yang dikeluarkan tiap kali makan

[7] Selisih harga per hari merupakan selisih biaya makat dalam satu hari dengan biaya harapan yang dikeluarkan dalam sehari

[8] Fragmen *fitness* merupakan nilai sub *fitness* yang digunakan untuk proses ekstraksi vaksin. Nilai ini didapatkan dengan menggunakan persamaan berikut,

$$f(x) = \frac{1}{p+1} + \frac{1}{l+1} + \frac{1}{k+1} + \frac{1000}{b+1}$$

Dimana :

p = selisih protein per makan

l = selisih lemak per makan

k = selisih karbohidrat per makan

b = selisih biaya per makan

contoh perhitungan fragmen *fitness* untuk hari pertama makan pagi =

$$f(x) = \frac{1}{87{,}12+1} + \frac{1}{211{,}88+1} + \frac{1}{1257{,}3+1} + \frac{1000}{98200+1} = 0{,}027023562$$

Untuk mendapatkan nilai kesamaan dalam satu kromosom digunakan cara berikut,

Kromosom	3	8	8	1	3	4	1	1	3	10	7	4	3	4	2	2	2	10	5	3	3	7	2	2
3	0				1				1				1				0				1			
8		0				0				0				0				0				0		
8			0				0				0				0				0				0	
1				0				1				0				0				0				0
3	1				0				1				1				0				1			
4		0				0				0				1				0				0		
1			0				0				0				0				0				0	
1				1				0				0				0				0				0
3	1				1				0				1				0				1			
10		0				0				0				0				1				0		
7			0				0				0				0				0				0	
4				0				0				0				0				0				0
3	1				1				1				0				0				1			
4		0				1				0				0				0				0		
2			0				0				0				0				0				1	
2				0				0				0				0				0				1
2	0				0				0				0				0				0			
10		0				0				1				0				0				0		
5			0				0				0				0				0				0	
3				0				0				0				0				0				0
3	1				1				1				1				0				0			
7		0				0				0				0				0				0		
2			0				0				0				1				0				0	
2				0				0				0				1				0				0

Jumlah = 30

Keterangan=

Nilai kesamaan dihitung jika kedua gen merupakan gen yang sejenis makanannya (makanan pokok dengan pokok, sayur dengan sayur, dst). Jika sama bernilai 1, jika berbeda bernilai 0. Pengecualian jika membandingkan dengan dirinya sendiri, maka nilainya 0. Nilai kesamaan merupakan jumlah angka 1 dalam tabel tersebut.

Setelah didapatkan nilai kesamaan, maka dapat dihitung nilai *fitness* untuk antibodi tersebut dengan menggunakan persamaan

$$f(x) = \frac{1}{P+1} + \frac{1}{L+1} + \frac{1}{K+1} + \frac{1000}{B+1} + \frac{1}{S+1}$$

dimana,

P = total selisih protein

L = total selisih lemak

K = total selisih biaya

S = jumlah nilai kesamaan

Sehingga untuk nilai *fitness* dari antibodi 1 ialah

$$f(x) = \frac{1}{269{,}48+1} + \frac{1}{625{,}1+1} + \frac{1}{2771{,}868+1} + \frac{1000}{708900+1} + \frac{1}{30+1}$$

$f(x) = 0{,}039323656$

Dengan cara yang sama dapat dihitung nilai *fitness* untuk antibodi lainnya, sehingga didapatkan tiap antibodi memiliki nilai *fitness* sebagai berikut,

Antibodi	Selisih harga	Protein	Lemak	Karbohidrat	Kesamaan	*Fitness*
1	708900	269,480	625,100	2771,868	30	0,039323656
2	518900	254,160	1100,900	5609,400	26	0,04396906
3	733600	2392,600	267,500	6359,140	24	0,045662544
4	602500	1231,776	950,000	4596,168	16	0,062563505
5	693700	2129,260	0,000	2469,920	28	1,036798436

A.6 Ekstraksi Vaksin

Urutkan antibodi berdasarkan nilai *fitness-nya*

antibodi	*fitness*
1	0,039323656
2	0,04396906
3	0,045662544
4	0,062563505
5	1,036798436

diurutkan →

antibodi	*fitness*
5	1,036798436
4	0,062563505
3	0,045662544
2	0,04396906
1	0,039323656

Pilih antibodi terbaik

Terbaik = Antibodi 5

Gen		1 9 5 2	2 4 5 2	3 7 7 4	2 5 6 1	2 7 7 4	2 9 8 2
Selisih	protein	929,38	101,72	0	301,62	169,22	958,26
	lemak	0	0	194,78	239,18	266,72	256,28
	karbohidrat	533,4	467,24	659,18	1377	998,8	0
	harga	65900	141900	113400	153300	113400	105800
fragment *fitness*		1,01812	1,01892	1,01544	0,01472	0,01943	1,01438

A.6.1 M1

Pilih satu fragmen secara roulette wheel

fragmen	*fitness*	probabilitas	rw
1	1,018120364	0,248261159	0,248261159
2	1,018918026	0,248455663	0,496716822
3	1,015440777	0,247607761	0,744324583
4	0,014716822	0,003588589	0,747913172
5	0,019428461	0,004737487	0,752650659
6	1,014380993	0,247349341	1
total	4,101005442		

r	0,465930526
terpilih	2

Dengan nilai probabilitas diambil dari

$p = \frac{f(x)_i}{\sum f(x)}$ dimana nilai f(x) ialah nilai *fitness* fragmen. Sedangkan nilai roulette wheel dihitung dengan $rw = \frac{p_i}{\sum_0^i p}$.

Dari nilai random, didapatkan fragmen yang terpilih ialah fragmen nomor 2. Pasangkan fragmen terpilih dengan fragmen lainnya, dan hitung *fitness-nya*.

fragmen		1	3	4	5	6
selisih	protein	929,38	0	301,62	169,22	958,26
	lemak	0	194,78	239,18	266,72	256,28
	karbohidrat	533,4	659,18	1377	998,8	0
	harga	65900	113400	153300	113400	105800
pasangan		2-1	2-3	2-4	2-5	2-6
selisih	protein	1031,1	101,72	403,34	270,94	1059,98
	lemak	0	194,78	239,18	266,72	256,28
	karbohidrat	1000,64	1126,42	1844,24	1466,04	467,24
	harga	207800	255300	295200	255300	247700
fitness pasangan		1,00678	0,01965	0,01057	0,01201	0,01100

Selisih protein, lemak, karbohidrat, serta biaya untuk tiap pasangan fragmen merupakan jumlah selisih dari masing masing pasangan fragmen. Misal pasangan 2-1, selisih protein = 101,72 + 929,38 = 1031,1

Pilih pasangan dengan nilai *fitness* tertinggi sebagai vaksin M1

Fragmen 1-2

Vaksin 1	1	9	5	2	2	4	5	2														

Masukkan vaksin baru kedalam memori vaksin

Karena memori vaksin masih kosong, maka vaksin akan alangsung dimasukkan.

A.6.2 M2

Tentukan panjang vaksin M2 sesuai dengan persamaan

$B_l = 2 + l$

$$l = \begin{cases} 0, & r \leq 0.6 \\ 1, & 0.6 < r \leq 0.85 \\ 2, & 0.85 < r \leq 0.95 \\ 3, & 0.95 < r \end{cases}$$

Dimana r ialah nilai random dari 0 sampai 1.

r	0,752131727
l	1
B_l	3

Buat pasangan fragmen secara berurutan sepanjang B_l, dan hitung *fitness-nya*

pasangan	123	234	345	456
protein	1031,1	403,34	470,84	1429,1
lemak	194,78	433,96	700,68	762,18
karbohidrat	1659,82	2503,42	3034,98	2375,8
harga	321200	408600	380100	372500
fitness	0,0097921	0,007618897	0,006504776	0,005114849

Ambil pasangan dengan nilai *fitness* tertinggi sebagai kandidat vaksin jenis M2.

Fragmen 1-2-3

Vaksin 2	1	9	5	2	2	4	5	2	3	7	7	4											

Masukkan vaksin baru kedalam memori vaksin, sehinga memori vaksin saat ini berisi

Vaksin 1	1	9	5	2	2	4	5	2															
Vaksin 2	1	9	5	2	2	4	5	2	3	7	7	4											

Tetapi dikarenakan isi memori vaksin saat ini lebih dari kapasitas memori vaksin seharusnya (ditentukan 10% dari populasi, yaitu 10% * 5 = 0,5 ≈ 1), maka buang vaksin dengan nilai *fitness* terendah hingga sesuai dengan kapasitas memori vaksin.

Vaksin 1	1,00678
Vaksin 2	0,009792

Dari vaksin diatas, terlihat vaksin 2 lebih rendah nilai *fitness-nya*, maka buang vaksin 2. Sehingga kondisi memori vaksin saat ini menjadi

Vaksin 1	1	9	5	2	2	4	5	2															

A.7 *Crossover*

Pilih nilai random untuk tiap antibodi, jika nilai tersebut lebih rendah dari parameter probabilitas *crossover*, maka antibodi tersebut akan menjadi kandidat untuk proses *crossover*.

probabilitas *crossover*		**0,4**
antibodi	**r**	**Cross**
1	0,626752101	False
2	0,302460231	True
3	0,902918613	False
4	0,126084061	True
5	0,48164155	False

Jika ternyata didapati kandidat *crossover* berjumlah ganjil, maka pilih satu lagi antibodi secara random. Pilih titik *crossover* secara random sesuai panjang kromosom, lalu lakukan proses *crossover* terhadap setiap pasangan kandidat.

Parent = 2 & 4, cross *point* = 10

A2	3	4	9	2	1	10	4	1	1	8	6	4	1	10	4	4	2	10	8	1	3	7	9	4

A4	2	8	7	3	2	10	1	1	3	2	2	2	1	5	7	4	1	4	5	1	1	11	2	2

Offspring

OC1	3	4	9	2	1	10	4	1	1	2	2	2	1	5	7	4	1	4	5	1	1	11	2	2
OC2	2	8	7	3	2	10	1	1	3	8	6	4	1	10	4	4	2	10	8	1	3	7	9	4

A.8 Mutasi

Pilih nilai random untuk tiap antibodi, jika nilai tersebut lebih rendah dari parameter probabilitas mutasi, maka antibodi tersebut akan menjadi kandidat untuk proses mutasi.

mutation probability		**0,1**
antibodi	**r**	**Mutate**
1	0,604832815	FALSE
2	0,806990298	FALSE
3	0,680817496	FALSE
4	0,024515383	TRUE
5	0,618989305	FALSE

Pilih titik mutasi, lalu lakukan proses mutasi pada antibodi terpilih. Mutasi dilakukan dengan mengubah gen dengan nilai random yamg masih sesuai dengan jarak nilai yang diperbolehkan pada lokasi tersebut.

Parent = 4, *mutation point* = 4

A4	2	8	7	3	2	10	1	1	3	2	2	2	1	5	7	4	1	4	5	1	1	11	2	2

Offspring

OM1	2	8	7	2	2	10	1	1	3	2	2	2	1	5	7	4	1	4	5	1	1	11	2	2

Pada contoh terlihat titik mutasi berada pada nomor 4, yang mana titik tersebut ialah lokasi untuk menu minuman. Menu minuman sesuai daftar memiliki maksimal jumlah nilai 4, sehingga nilai random dipilih antara nilai minimum (1) hingga maksimum (4).

A.9 Vaksinasi

Piilh satu antibodi secara random untuk divaksinasi.

r	1

Terpilih Antibodi 1

Pilih satu vaksin secara random dari memori vaksin

r	1

Terpilih vaksin 1

Proses vaksinasi akan menggantikan gen gen pada antibodi terpilih sesuai dengan gen pada vaksin

A1	3	8	8	1	3	4	1	1	3	10	7	4	3	4	2	2	2	10	5	3	3	7	2	2
V1	1	9	5	2	2	4	5	2																

Hasil vaksinasi

AV1	1	9	5	2	2	4	5	2	3	10	7	4	3	4	2	2	2	10	5	3	3	7	2	2

Hasil vaksinasi akan dihitung kembali nilai *fitness-nya*, jika ternyata nilai *fitness-nya* membaik, maka hasil vaksinasi akan menggantikan antibodi aslinya, tetapi jika nilai *fitness-nya* memburuk, maka hasil vaksinasi akan dibuang.

Fitness hasil vaksinasi	1,0354291
Fitness parent	0,039323656

Dikarenakan nilai *fitness* yang membaik, maka hasil vaksinasi akan menggantikan antibodi aslinya.

A.10 Perhitungan nilai *fitness*

Setelah melalui proses *Crossover*, mutasi, dan vaksinasi, maka kondisi dan jumlah antibodi dalam populasi akan mengalami perubahan. Oleh karena itu dilakukan perhitungan nilai *fitness* kembali.

Kondisi populasi,

A1	AV1	1	9	5	2	2	4	5	2	3	10	7	4	3	4	2	2	2	10	5	3	3	7	2	2
A2	A2	3	4	9	2	1	10	4	1	1	8	6	4	1	10	4	4	2	10	8	1	3	7	9	4
A3	A3	2	8	8	4	1	9	6	1	2	5	6	3	3	2	8	3	2	2	7	3	3	7	7	4
A4	A4	2	8	7	3	2	10	1	1	3	2	2	2	1	5	7	4	1	4	5	1	1	11	2	2
A5	A5	1	9	5	2	2	4	5	2	3	7	7	4	2	5	6	1	2	7	7	4	2	9	8	2
A6	OC1	3	4	9	2	1	10	4	1	1	8	2	2	1	5	7	4	1	4	5	1	1	11	2	2
A7	OC2	2	8	7	3	2	10	1	1	3	2	6	4	1	10	4	4	2	10	8	1	3	7	9	4
A8	OM1	2	8	7	2	2	10	1	1	3	2	2	2	1	5	7	4	1	4	5	1	1	11	2	2

Setelah menghitung nilai *fitness* seperti pada poin A.5, didapatkan nilai *fitness* untuk masing-masing antibodi ialah sebagai berikut,

antibodi	*fitness*
1	1,0354291
2	0,04396906
3	0,045662544
4	0,062563505
5	1,0354291
6	0,034727451
7	0,045147669
8	0,051992069

A.11 Ekstraksi imun

Proses ekstraksi imun berfungsi untuk menyimpan antibodi terbaik agar tidak terbuang secara tidak sengaja. Proses ini menyimpan 10 % antibodi terbaik dari populasi.

jumlah kandidat 10 % * 8 = 0,8 = 1

antibodi	*fitness*

antibodi	*fitness*

1	1,035429
2	0,043969
3	0,045663
4	0,062564
5	1,035429
6	0,034727
7	0,045148
8	0,051992

diurutkan -->

5	1,035429
1	1,035429
4	0,062564
8	0,051992
3	0,045663
7	0,045148
2	0,043969
6	0,034727

Terpilih = Antibodi 5

Karena memori imun masih kosong, langsung masukkan antibodi terpilih kedalam memori imun

Memori imun

A5	1	9	5	2	2	4	5	2	3	7	7	4	2	5	6	1	2	7	7	4	2	9	8	2

A.12 Seleksi Imun

Hitung nilai kedekatan antar antibodi (*affinity*) dengan persamaan

$$A_{ij} = \frac{1}{1 + \sqrt{\sum_{v=1}^{m}(i_v - j_v)^2}}$$

	1	2	3	4	5	6	7	8
1	1	0,0650	0,0647	0,0597	0,0753	0,0637	0,0608	0,0598
2	0,0650	1	0,0667	0,0601	0,0824	0,0707	0,1054	0,0602
3	0,0647	0,0667	1	0,0798	0,0909	0,0804	0,0664	0,0790
4	0,0597	0,0601	0,0798	1	0,0821	0,1054	0,0707	0,5000
5	0,0753	0,0824	0,0909	0,0821	1	0,0748	0,0926	0,0824
6	0,0637	0,0707	0,0804	0,1054	0,0748	1	0,0601	0,1061
7	0,0608	0,1054	0,0664	0,0707	0,0926	0,0601	1	0,0705
8	0,0598	0,0602	0,0790	0,5000	0,0824	0,1061	0,0705	1

Contoh perhitungan kedekatan antibodi 1 dengan antibodi 2

$$A_{12} = \frac{1}{1 + \sqrt{(3-3)^2 + (8-4)^2 + (8-9)^2 + (1-2)^2 + (3-1)^2 + (4-10)^2 + \cdots + (2-4)^2}}$$

$$A_{12} = 0{,}064987838 \approx 0{,}0650$$

Setelah didapatkan nilai *affinity* untuk masing-masing antibodi, maka selanjutnya dapat dicari nilai konsentrasinya,

$$C_i = \frac{banyak\ antibodi\ yang\ nilai\ kedekatannya\ terhadap\ antibodi\ i\ lebih\ besar\ daripada\ \phi}{N}$$

Dengan ϕ adalah konstanta *similarity* dan N ialah banyak antibodi dalam populasi

antibodi	konsentrasi	Probabilitas konsentrasi
1	0,125	0,125

2	0,125	0,125
3	0,125	0,125
4	0,125	0,125
5	0,125	0,125
6	0,125	0,125
7	0,125	0,125
8	0,125	0,125
sum	1	

Dengan nilai $\phi = 0,9$ maka untuk mencari nilai konsentrasi antibodi 1 ialah sebagai berikut,

$$C_1 = \frac{1}{8} = 0,125$$

Probabilitas berdasarkan konsentrasi dapat dicari dengan

$$P_d = \frac{c_i}{\sum_{i=1}^{N} c_i}$$

Sehingga untuk mendapatkan nilai probabilitas konsentrasi antibodi 1 dapat dihitung dengan,

$$P_d = \frac{0,125}{0,125 + 0,125 + 0,125 + 0,125 + \cdots + 0,125}$$

$$P_d = \frac{0,125}{1} = 0,125$$

Hitung nilai probabilitas seleksi berdasarkan probabilitas dari *fitness* dan konsentrasi,

$$P = \mu P_f + (1 - \mu) P_d$$

Dimana μ adalah konstanta *affinity*, Pf merupakan probabilitas berdasarkan nilai *fitness* yang dapat dicari dengan

$$P_f = \frac{f_i}{\sum_{i=1}^{N} f_i}$$

dan Pd merupakan probabilitas berdasarkan konsentrasi.

antibodi	*fitness*	*Fitness based probability*	*Concentration based probability*	*survival Probability*
1	1,0354291	0,439687497	0,125	0,219406249
2	0,04396906	0,018671144	0,125	0,093101343
3	0,045662544	0,01939027	0,125	0,093317081
4	0,062563505	0,026567141	0,125	0,095470142
5	1,0354291	0,439687497	0,125	0,219406249
6	0,034727451	0,014746762	0,125	0,091924028
7	0,045147669	0,019171632	0,125	0,09325149
8	0,051992069	0,022078057	0,125	0,094123417
sum	2,3549205			

Jika nilai $\mu = 0{,}3$,untuk mendapatkan nilai probabilitas bertahan hidup dari antibodi 1, dapat dihitung dengan

$$P_{f1} = \frac{1{,}0354291}{1{,}0354291 + \ 0{,}04396906 + \ 0{,}045662544 + \cdots + 0{,}051992069}$$

$$P_{f1} = \frac{1{,}0354291}{2{,}3549205} = \ 0{,}439687497$$

$$P = \ \mu P_f + (1 - \mu) P_d$$

$$P = (0{,}3 * 0{,}439687497) + ((1 - 0{,}3) * 0{,}125)$$

$$P = (0{,}3 * 0{,}439687497) + (0{,}7 * 0{,}125)$$

$$P = 0{,}219406249$$

Selanjutnya hitung berapa banyak antibodi yang akan mati maupun yang dapat bertahan hidup menuju generasi selanjutnya dengan persamaan

$$N_r = \ N - N_d$$

Dimana Nr ialah jumlah antibodi yang akan bertahan ke generasi selanjutnya, N merupakan jumlah populasi saat ini, Nd adalah jumlah antibodi yang akan mati yang dihitung dengan

$$N_d = N\sigma PD$$

σ adalah konstanta *death factor*, sedangkan nilai PD dapat dicari dengan persamaan

$$PD = \frac{f_{avg}}{f_{max}}$$

Dengan nilai $\sigma = 0{,}5$,perhitungan untuk mendapatkan banyak generasi yang akan bertahan akan nejadi seperti berikut,

antibodi	*fitness*
1	1,0354291
2	0,04396906
3	0,045662544
4	0,062563505
5	1,0354291
6	0,034727451
7	0,045147669
8	0,051992069
avg	0,294365062
max	1,0354291

$$PD = \frac{f_{avg}}{f_{max}} = \frac{0{,}294365062}{1{,}0354291}$$

$$PD = \ 0{,}284292823$$

$$N_d = N\sigma PD$$

$$N_d = 8 * 0{,}5 * 0{,}284292823$$

$N_d = 1{,}137171292 \approx 1$

$N_r = N - N_d$

$N_r = 8 - 1$

$N_r = 7$

Setelah mendapatkan berapa banyak antibodi yang dapat bertahan hidup, maka urutkan antibodi dalam populasi berdasarkan nilai probabilitas bertahan hidup, ambil sebanyak Nr antibodi terbaik berdasarkan nilai probabilitas tersebut.

antibody	*probability*	*fitness*
1	0,0961819	1,0354291
2	0,097207515	0,04396906
3	0,097581404	0,045662544
4	0,101312808	0,062563505
5	0,316102656	1,0354291
6	0,095167147	0,034727451
7	0,097467729	0,045147669
8	0,09897884	0,051992069

diurutkan -->

antibody	*probability*	*fitness*	
5	0,316102656	1,0354291	
1	0,0961819	1,0354291	
4	0,101312808	0,062563505	
8	0,09897884	0,051992069	
3	0,097581404	0,045662544	
7	0,097467729	0,045147669	
2	0,097207515	0,04396906	
6	0,095167147	0,034727451	mati

Terlihat yang dapat melanjutkan ke generasi selanjutnya ialah antibodi 1,3,4,5,7,8. Antibodi 2 dan 6 berada pada urutan terbawah diluar nilai Nr sehingga antibodi tersebut dianggap mati dan dikeluarkan dari populasi.

Dari hasil seleksi, maka populasi baru saat ini menjadi seperti berikut,

A1	1	9	5	2	2	4	5	2	3	10	7	4	3	4	2	2	2	10	5	3	3	7	2	2
A2	3	4	9	2	1	10	4	1	1	8	6	4	1	10	4	4	2	10	8	1	3	7	9	4
A3	2	8	8	4	1	9	6	1	2	5	6	3	3	2	8	3	2	2	7	3	3	7	7	4
A4	2	8	7	3	2	10	1	1	3	2	2	2	1	5	7	4	1	4	5	1	1	11	2	2
A5	1	9	5	2	2	10	1	1	3	7	7	4	1	5	7	4	2	7	7	4	2	9	8	2
A7	2	8	7	3	2	10	1	1	3	2	6	4	1	10	4	4	2	10	8	1	3	7	9	4
A8	2	8	7	2	2	10	1	1	3	2	2	2	1	5	7	4	1	4	5	1	1	11	2	2

A.13 Akhir iterasi

Pada akhir iterasi, diambil antibodi dengan nilai *fitness* tertinggi sebagai hasil dari algoritma.

antibodi	*fitness*
1	1,0354291
2	0,04396906
3	0,045662544
4	0,062563505
5	1,0354291
7	0,045147669
8	0,051992069

Dari antibodi dalam populasi, yang memiliki nilai *fitness* terbaik ialah antibodi 5 dan 1, karena terdapat 2 hasil, maka pilih salah satu sebagai hasil dari algoritma

Contoh Hasil Menu

1	1	9	5	2	2	4	5	2	3	10	7	4	3	4	2	2	2	10	5	3	3	7	2	2
Nama	Nasi	Tumis kacang panjang	Telur dadar	Sirup	Nasi Pecel	mie telur	Telur dadar	Sirup	Nasi Goreng	Tumis kangkung	tempe bacem	Teh	Nasi Goreng	mie telur	Dadar jagung	Sirup	Nasi Pecel	Tumis kangkung	Telur dadar	Susu	Nasi Goreng	Sayur bayam	Dadar jagung	Sirup
Protein	101,08	1182,56	246,24	0	300,2	275,88	246,24	0	356,82	104,88	141,74	0	356,82	275,88	71,06	0	300,2	104,88	246,24	298,68	356,82	109,44	71,06	0
Lemak	10,64	467,78	278,54	0,76	425,6	125,78	278,54	0,76	352,64	357,2	117,8	0	352,64	125,78	205,2	0,76	425,6	357,2	278,54	301,34	352,64	59,28	205,2	0,76
Karbohidrat	1060,2	321,86	15,96	104 1,58	1447,42	152 9,88	15,96	104 1,58	119 2,44	163,78	84,74	699,2	119 2,44	152 9,88	359,1	104 1,58	1447,42	163,78	15,96	419,14	119 2,44	337,44	359,1	104 1,58
Harga	0	38000	28500	11400	95000	57000	28500	11400	95000	38000	19000	11400	95000	57000	11400	11400	95000	38000	28500	11400	95000	38000	11400	11400
Energi	474 0,88	102 27,7	355 5,66	417 3,16	1082 0,88	835 5,06	355 5,66	417 3,16	937 0,8	428 9,44	196 6,12	279 6,8	937 0,8	835 5,06	356 7,44	417 3,16	1082 0,88	428 9,44	355 5,66	558 3,34	937 0,8	232 1,04	356 7,44	417 3,16
per makan				22697,4				26904,76				18423,16				25466,46				24249,32				19432,44
per hari												68025,32												69148,22
total energi																								137174

LAMPIRAN B HASIL WAWANCARA

B.1 Wawancara di panti asuhan

Tempat : Panti asuhan Nurul Abyadh, jalan Bendungan Sigura-gura 1 no. 8 Malang

Narasumber : Ibu pengurus panti asuhan

1. Tanya : Berapa banyak anak yang menghuni panti asuhan ?

   Jawab : 39 anak dengan rincian sebagai berikut,

No	Nama	Jenis Kelamin	Umur	Tinggi Badan	Berat Badan
1	Aldi	L	10	145	30
2	Firman	L	11	135	28
3	Yohanes	L	12	163	52
4	Irman	L	13	135	30
5	Misbakhul Ulum	L	15	160	38
6	Abd Kodir F	L	15	156	42
7	Ikbal Hobir	L	16	159	41
8	Gunawan Putra	L	16	170	54
9	Tio	L	16	158	58
10	Sam'an	L	17	170	55
11	Achmad Budiono	L	17	155	42
12	Ainur Rahman	L	17	162	50
13	Susanto	L	17	158	50
14	Jonathan	L	17	169	61
15	Wardi	L	18	165	50
16	Jonathan Firdaus	L	19	168	50
17	Novi	P	9	125	38
18	Sela Indriani	P	11	132	27
19	Srilati	P	11	141	38
20	Dinda P.P	P	12	135	30
21	Vanisa Ayu Maharani	P	12	135	30
22	Azyarba B Oktaviana	P	13	163	35
23	Urwatul Wasikoh	P	13	145	39
24	Miftahul Jannah	P	14	153	39
25	Raudhatul Jannah	P	14	145	39
26	Nisfatul Laely	P	14	155	50
27	Siti Parihah	P	15	149	40
28	Selvi L.L	P	15	156	55
29	Rohmadiyah	P	15	153	47
30	Insiyah	P	16	150	64

31	Khoirotun Nisa'	P	16	159	52
32	Siti Asih	P	16	155	50
33	Indah Sosilowati	P	16	157	61
34	Eni	P	17	160	56
35	Frida Sucila	P	17	0	40
36	Ari Qurrota A'yuina	P	17	155	50
37	Mina	P	17	158	45
38	Nunung	P	17	150	39
39	Malisa	P	18	149	41

2. Tanya : Bagaimana pengaturan menu makan di panti asuhan saat ini ?

   Jawab : Menu makanan ditentukan untuk 1 minggu kedepan dengan acuan yang penting ada nasi, sayur, lauk, minuman (4 sehat 5 sempurna), hanya saja kalau susu tidak bisa setiap hari karena memang biayanya tinggi.

3. Tanya : Bagaimana pengaturan anggaran belanja di panti asuhan ?

   Jawab : Untuk belanja tergantung menu yang akan dibuat hari itu. Sehari belanja ke pasar dua kali, pagi dan siang. Pagi untuk makan pagi, dan siang untuk makan siang dan malam. Sekali belanja bisa mendapatkan anggaran minimal Rp 150.000. Tetapi untuk beras dan air minum tidak beli karena dari sumbangan sudah mencukupi.

B.2 Wawancara dengan ahli gizi

Tempat : Fakultas kedokteran Universitas Brawijaya

Narasumber : Ibu Laksmi Karunia Tanuwijaya, S.Gz., Mbiomed

1. Tanya :Bagaimana cara menghitung kebutuhan gizi untuk anak panti asuhan ?

   Jawab : Untuk memudahkan proses penghitungan, kebutuhan gizi untuk anak panti asuhan dapat menggunakan tabel AKG sesuai dengan jenis kelamin dan kelompok umur dengan mengabaikan faktor tinggi badan dan berat badan. Untuk pembagian 3 kali makan bisa diterapkan aturan pembagian 20-30-20 atau 25-30-25, yaitu 20% pagi, 30% siang, 20% malam atau 25% pagi, 30% siang, 25 % malam. Sisa 20-30% gizi yang dibutuhkan didapatkan dari makanan ringan yang dapat diabaikan untuk penelitian ini. Selain itu masih ada 10% toleransi lebih atau kurangnya kandungan gizi terhadap kebutuhan anak panti.

2. Tanya : Bagaimana cara mengetahui kandungan gizi suatu makanan ?

   Jawab : Untuk mengetahui kandungan gizi makanan dapat melihat referensi daftar komposisi bahan makanan. Dalam daftar tersebut sudah disebutkan berapa banyak kandungan gizi suatu makanan beserta porsinya secara umum.