OTOMATISASI DESAIN *TEST CASE* PENGUJIAN PERANGKAT LUNAK METODE *BLACK-BOX TESTING* DENGAN TEKNIK *EQUIVALENCE PARTITIONING* MENGGUNAKAN ALGORITMA GENETIKA

Arochman[1]
[1]*Program Studi Teknik Informatika,STMIK Widya Pratama*
Jl. Patriot 25 Pekalongan Telp (0285)427816
email : arochman.aryanta@gmail.com
Arief Soma Darmwan[2]
[1]*Program Studi Teknik Informatika,STMIK Widya Pratama*
Jl. Patriot 25 Pekalongan Telp (0285)427816
Fx Heru Santoso
[1]*Program Studi Teknik Informatika,STMIK Widya Pratama*
Jl. Patriot 25 Pekalongan Telp (0285)427816

ABSTRAK

Pengujian *black-box* sangat penting karena teknik tersebut mampu mengidentifikasi kesalahan dalam fungsi, antar muka, model data, dan akses kesumber data *eksternal*. Dalam pelaksanaan sering timbul masalah penguji tidak perna yakin apakah perangkat lunak yang diuji telah benar-benar lolos dalam pengujian. Hal ini terjadi karena kemungkinan masih ada beberapa jalur eksekusi yang belum perna teruji. Penguji seharusnya membuat setiap kemungkinan kombinasi data masukan untuk pengujian. Pemilihan data masukan untuk menemukan kesalahan menjadi masalah bagi penguji karena memiliki probabilitas yang tinggi, sehingga desain *test case* secara otomatis dapat menjadi solusi. Untuk menghasilkan desain *test case* secara otomatis dalam pengujian perangkat lunak metode *black-box* dengan teknik *equivalence partitioning* dibutuhkan sebuah teknik *artificial intelligence. Artificial intelligence* yang digunakan dalam optimasi dan efisiensi desain *test case* adalah algoritma genetika. Algoritma genetika adalah suatu algoritma pencarian yang berbasis pada mekanisme seleksi alam dan genetika. Terkait dengan pengujian *black-box* algoritma genetika dapat digunakan dalam pencarian kemungkinan-kemungkinan data masukan untuk desain *test case* secara otomatis.

Kata Kunci : algoritma genetika, *equivalence partitioning, black-box testing*

1. PENDAHULUAN

1.1 Latar Belakang

Pengujian adalah serangkaian kegiatan yang dapat direncanakan sebelumnya dan dilakukan secara sistematis. Kegiatan yang terkait dengan pengujian meliputi proses menganalisis item dan program serta fitur dari item perangkat lunak (Agarwal, B. B.; Tayal, S.P.; Gupta, M., 2010). Tujuan Pengujian perangkat lunak adalah mendeteksi perbedaan antara hasil keluaran perangkat lunak dengan kondisi yang diharapkan dan menemukan kesalahan. Dengan pengujian kualiatas dan kepercayaan dalam berfungsinya perangkat lunak akan meningkat.

Menurut Bortolino pada penelitianya *"Software Testing Research: Achievments, Challenges, Dream"* menyebutkan beberapa pendekatan yang dapat digunakan dalam pengujian perangkat lunak antara lain, pendekatan model, pendekatan teknik, pendekatan berbasis pencarian untuk generasi masukan uji dan pendekatan penilaian kinerja atribut (Bertolino, Antonia, 2007). Salah satu metode yang sering digunakan adalah *black-box testing*. Pengujian *black-box* juga disebut pengujian perilaku, berfokus pada fungsional dan spesifikasi perangkat lunak (Agarwal, B. B.; Tayal, S.P.; Gupta, M., 2010). Dalam pengujian *black-box,* penguji hanya tahu masukan yang dapat diberikan kepada sistem dan keluaran yang harus dihasilkan. Kelebihan dari pengujian ini antara lain desainer dan penguji yang tidak mengikat tehadap satu sama lain, penguji tidak perlu tahu dari setiap bahasa pemrograman tertentu, pengujian dilakukan dari sudut pandang pengguna bukan desainer.

Pengujian *black-box* sangat penting karena teknik tersebut mampu mengidentifikasi kesalahan dalam fungsi, antar muka, model data, dan akses ke sumber data *eksternal*.

Dalam pelaksanaan sering timbul masalah penguji tidak perna yakin apakah perangkat lunak yang diuji telah benar-benar lolos dalam pengujian. Hal ini terjadi karena kemungkinan masih ada beberapa jalur eksekusi yang belum perna teruji. Penguji seharusnya membuat setiap kemungkinan kombinasi data masukan untuk pengujian (Hendraputra, Ade; Pratondo, Agus; wijaya, Dedy Rahman; dkk;, 2009). Pemilihan data masukan untuk menemukan kesalahan menjadi masalah bagi penguji karena memiliki probabilitas yang tinggi (Agarwal, B. B.; Tayal, S.P.; Gupta, M., 2010).

Tingginya probabilitas data masukan pada pengujian *black-box,* para vendor mengalami kesulitan dalam merancang *test case* pengujian. Menurut Jason Bau pada penelitianya "*State of the Art Automated Black-Box Web Application Vulnerability Testing",* Hal tersebut merupakan salah satu faktor yang mengakibatkan tingginya kerentanan dalam berbagai aplikasi web (Bau, Jason; Bursztein, Elie; Gupta, Divij; Mitchell, John, 2010).

Dalam pengujian *black-box* terdapat tiga motode antara lain: *Equivalence Partitioning, Boundary value testing, Use case testing*. Dari tiga motode tersebut panguji harus mampu medesain *test case* yang di mungkin dengan tepat. Secara konvensional, medesain *test case* untuk menetukan keluaran yang benar adalah pekerjaan yang banyak membutuhkan waktu sehingga kurang efisien (Dick, S.; Kandel, A., 2005).

Medesain *test case* secara otomatis dapat menjadi solusi. Untuk menghasilkan *test case* secara otomatis dibutuhkan sebuah teknik *Artificial Intelligence* (Dick, S.; Kandel, A., 2005). *Artificial Intelligence* digunakan untuk membuat formulasi dan optimasi pemilihan data dalam medesain *test case*. Adapun *Artificial Intelligence* yang dapat digunakan antara lain: Fuzzy logic, Neural network, Algoritma genetika (*genetic algorithm)*, Case Based Reasoning dan algoritma yang lain.

Dari beberapa komputasi cerdas tersebut yang sangat tepat dalam penyelesaian masalah optimasi kompleks, yang sulit dilakukan oleh metode konvensional adalah algoritma genetika. Algoritma genetika adalah suatu algoritma pencarian yang berbasis pada mekanisme seleksi alam dan genetika (Desiani, Anita; Arhami, Muhammad, 2006). Terkait dengan pengujian *black-box,* algoritma genetika dapat digunakan dalam pencarian kemungkinan-kemungkinan data masukan dalam medesain *test case* dengan teknik *Equivalence Partitioning* yang optimal.

Penelitian terkait perna dilakukan oleh Lionel C. Briand dengan menggunakan algoritma C4.5 dapat membantu menganalisis kelemehan dan redudansi spesifikasi pengujian perangkat lunak (Briand, Lionel C.; Labiche, Yvan; Bawar, Zaheer, 2008). Selain itu pada tahun 2010 Abdul rauf mampu menunjukan optimasi *test case* dalam pengujian *Graphical User Interface (GUI)* dengan menggunakan algoritma genetika (Rauf, Abdul; Anwa, Sajid; Jaffer, M. Arfan; Shahid, Arshad Ali, 2010).

Dengan pertimbangan bahwa algoritma genetika mampu mengoptimasi desain *test case* pada pengujian GUI, maka akan dipilih algoritma genetika untuk optimasi dan otomatisasi desain *test case* pada pengujian perangkat lunak metode *black-box testing* dengan teknik *equivalence partitioning*.

1.2 Landasan Teori

1. Tinjauan Studi (Related Research)

Penelitian yang dilakukan oleh Lionel C. Briand, Yvan Labiche, and Zaheer Bawar pada tahun 2008. Permasalahan dalam penelitian tersebut antara lain, keterbatasan *tester* dalam memahami *test suite* dan adanya redudansi *test suite* pada pengujian *black-box* untuk perangkat lunak *open source.* Lionel menggunakan metode algoritma C4.5 dalam mengatasi permasahan diatas. Dengan pendekatan sebagai berikut: mengidentifikasi masalah pada keputusan C4.5 dan menghubungkannya dengan *test suite* potensial atau CP spesifikasi kekurangan. Kemudian menambah strategi untuk *test suite*. Hasil penelitian Lionel yang tersebut diatas adalah Algoritma C4.5 membantu *tester* perangkat lunak menganalisis kelemahan dan redundansi spesifikasi pengujian dan *test suite* dan iteratif diperbaiki. Hasil analisis menunjukan pengurangan *test suit* mencapai 50% (Briand, Lionel C.; Labiche, Yvan; Bawar, Zaheer, 2008).

Berikutnnya penelitian yang dilakukan oleh Abdul Rauf, Sajid Anwa, M. Arfan Jaffer, and Arshad Ali Shahid pada tahun 2010. Abdul rauf menyatakan bahwa setiap pengembang

perangkat lunak selalu ingin untuk menguji perangkat lunak secara menyeluruh untuk mendapatkan kepercayaan yang maksimal tentang kualitas. Tapi ini memerlukan upaya besar untuk menguji GUI karena kompleksitas yang terlibat dalam aplikasi tersebut. Abdul Rauf menggunakan Algoritma genetika dalam mengatasi permasalahannya. Dengan algoritma genetika dilakukan pencarian kombinasi tes yang sebaik mungkin dengan parameter beberapa kriteria uji yang telah ditetapkan. Hasil penelitian Abdul Rauf yang tersebut diatas adalah Algoritma genetika mampu mengoptimasi "fungsi cakupan" yang memenuhi kriteria tes atau pengujian. Analisis cakupan uji funsi *fitness* menunjukan bahwa sistem mampu mencapai lebih dari 85% cakupan (Rauf, Abdul; Anwa, Sajid; Jaffer, M. Arfan; Shahid, Arshad Ali, 2010).

2. Tinjauan Pustaka

Pengujian *Black-box*

Pengujian *Black-box* merupakan *strategi testing* di mana hanya memperhatikan atau memfokuskan kepada faktor fungsionalitas dan spesifikasi perangkat lunak (Hendraputra, Ade; Pratondo, Agus; wijaya, Dedy Rahman; dkk;, 2009). Pengujian *black-box* mengidenatifikasi jenis kesalahan antara lain kesalahan suatu fungsi, kesalahan suatu antar muka, kesalahan dalam pemodelan data dan kesalahan dalam akses ke sumber data *eksternal* (Agarwal, B. B.; Tayal, S.P.; Gupta, M., 2010).

Pengujian *black-box* sangat diperlukan dalam meningkatkan kualitas dan kepercayaan dari suatu perangkat lunak. Semakin besar tingkat kompleksitas suatu perangkat lunak akan membutuhkan pengujian yang semakin luas. Hal ini terlihat pada proses pemilihan jenis *input* yang dimungkinkan dalam pengujian.

Metode dalam pengujian *black-box* meliputi (Hendraputra, Ade; Pratondo, Agus; wijaya, Dedy Rahman; dkk;, 2009):

a. *Equivalence partitioning*

Merupakan teknik yang digunakan untuk mengurangi jumlah *test case* yang ada pada saat pengujian. Kasus uji yang didesain untuk *equivalence partitioning* berdasarkan pada evaluasi dari ekuivalensi jenis/*class* untuk kondisi *input*. *Class-class* yang ekuevalen mempresentasikan sekumpulan keadaan valid dan invalid untuk kondisi *input.* Biasanya kondisi *input* dapat berupa spesifikasi nilai numerik, kisaran nilai, kumpulan nilai yang berhubungan atau kondisi *boolean* (Hendraputra, Ade; Pratondo, Agus; wijaya, Dedy Rahman; dkk;, 2009).

b. *Boundary value testing*

Merupakan teknik desain *testing* yang paling dasar. Itu membantu *tester* untuk memilih subset yang kecil untuk membuat *test case* yang mungkin. Yang menjadi fokus pada teknik ini adalah batasan-batasan yang simple karena di situlah kebanyakan cacat pada perangkat lunak tersembuyi (Hendraputra, Ade; Pratondo, Agus; wijaya, Dedy Rahman; dkk;, 2009).

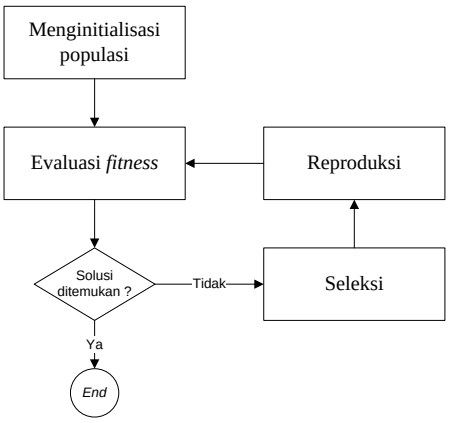
c. *Use case testing*

Dalam hal pengujian perangkat lunak, informasi yang ada pada *use case* sangat berguna bagi *tester.* Komponen utama dari pengujian dari suatu transaksi adalah data *testing.* Boris Beizer dalam bukunya menyarankan bahwa 30 sampai dengan 40 persen dari pengujian suatu transaksi adalah *generating, capturing, or extracting test data* (Hendraputra, Ade; Pratondo, Agus; wijaya, Dedy Rahman; dkk;, 2009).

Algoritma Genetika

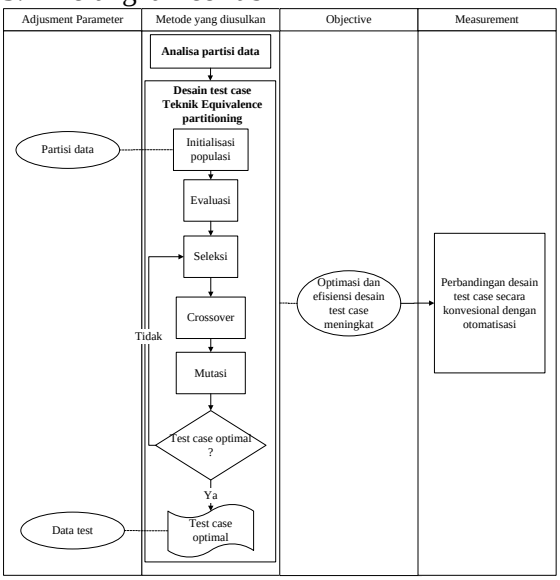
Algoritma Genetika adalah suatu algorima pencarian yang berbasi pada mekanisme seleksi alam dan genetika. Algoritma genetika merupakan salah satu algoritma yang sangat tepat digunakan dalam menyelesaikan masalah optimasi kompleks, yang sulit dilakukan oleh metode konvensional (Desiani, Anita; Arhami, Muhammad, 2006).

Dalam pemrograman, algoritma genetika diimplementasikan dengan menggunakan array yang berisi bit atau karakter untuk mewakili kromosom. *Flow chart* dari genetika algoritma dapat digambarkan sebagai berikut (Nedjah, Nadia; Abraham, Ajith; Mourelle, Luiza de Macedo, 2006).



Gambar 1. Siklus Algoritma Genetika

3. Kerangka Teoritis



Gambar 2. Kerangka Teoritis

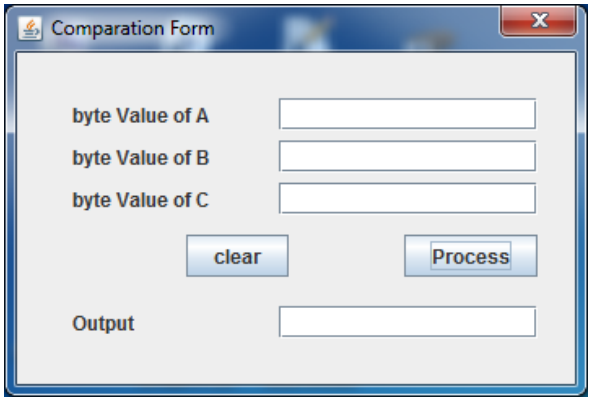
2. METODE PENELITIAN

2.1 Jenis Penelitian

Penelitian yang digunakan adalah *Eksperimen* dengan tujuan otomatisasi desain *test case* pada pengujian perangkat lunak metode *black-box testing* dengan teknik *equivalence partitioning* menggunakan algoritma genetika, sehingga lebih optimal dan efisien.

2.2 Populasi dan Sampel

Perancangan pengujian *black-box* yang akan dibahas sebagai studi kasus adalah sebuah aplikasi yang berfungsi untuk membandingkan 3 masukan yaitu a, b dan c. Adapun tampilan aplikasi tersebut adalah sebagai berikut:
tersebut adalah sebagai berikut:



Gambar 3. Tampilan aplikasi yang akan diuji

Logic program yang ada pada aplikasi diatas sebagai berikut:

```
private void comparation(byte a, byte b,
byte c){
        try{
                byte a =
        Byte.parseByte(ta.getText());
           byte b =
Byte.parseByte(tb.getText());
           byte c =
Byte.parseByte(tc.getText());
                if (a>b && a>c){

        to.setText(Byte.toString(a));
                }else if(b>a && b>c){
                        to.setText(Byte.toSt
                ring(b));
                }else if (c>a && c>b){
                        to.setText(Byte.toSt
        ring(c));
                }
        }catch(Exception e){
        to.setText("Data tidak sesuai");
        }
}
```

2.3 Metode Pengumpulan Data

Pada tahap persiapan dan penelitian awal, pelaksanaan pengumpulan data menggunakan jenis data sekunder. Data sekunder dikumpulkan dari beberapa buku, jurnal ilmiah dan website. Pengumpulan data sekunder dilakukan berdasarkan pemahaman yang lebih lengkap mengenai masalah pengujian *black-box*, algoritma genetika .

2.4 Teknik Pengolahan dan Analisa Data

1. Analisis data

Pengujian *black-box* hanya memfokuskan atau memperhatikan kepada faktor fungsionalitas (Agarwal, B. B.; Tayal, S.P.; Gupta, M., 2010) (Hendraputra, Ade; Pratondo, Agus; wijaya, Dedy Rahman; dkk;, 2009). Perancangan pengujian yang harus dilakukan adalah bagai mana membuat kombinasi masukan agar dapat melewati kondisi yang bernilai “true” dan bernilia “false”. Pada kode program 5.1

masukan bertipe byte, dalam bahasa pemrograman java tipe data byte memiliki range dari -127 sampai dengan 128 sehingga ada kemungkinan untuk melakukan kombinasi sebanyak 255 * 3 atau 765. Sebuah nilai kombinasi yang tinggi untuk memastikan semua seleksi terlewati.

Petujuk pelaksanaan dalam penyusunan *test case* dengan metode *equivalence partitioning*, adalah sebagai berikut (Romeo, 2003):

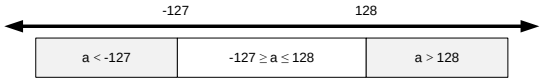
a. Jika masukan mempunyai jenjang tertentu, maka definisikan valid dan tak valid
b. Jika massukan membutuhkan nilai tertentu, definisikan kategori valid dan tidak valid.
c. Jika masukan membutuhkan himpunan masukan tertentu, definisikan kategori valid dan tidak valid.
d. Jika masukan adalah boolean, definisikan kategori valid dan tidak valid.

2. Desain Test Case dengan Teknik Equivalence Partitioning Secara Konvensional

a. Analisa partisi

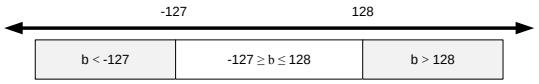
Partisi adalah sekumpulan nilai, yang dipilih dengan suatu cara dimana semua nilai di dalam partisi, diharapkan untuk diperlakukan dengan cara yang sama oleh komponen. Untuk fungsi **comparation** terdapat tiga masukan dan satu keluaran:

a. Partisi ekuivalensi untuk masukan "a"



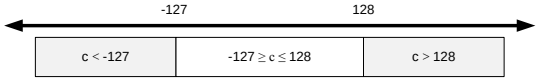
Gambar 4. Partisi ekuivalensi untuk masukan "a"

b. Partisi ekuivalensi untuk masukan "b"



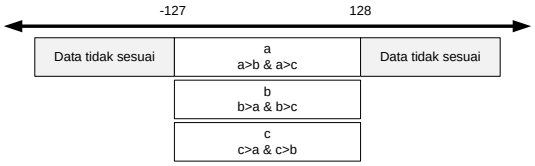
Gambar **Error! No text of specified style in document.**. Partisi ekuivalensi untuk masukan "b"

c. Partisi ekuivalensi untuk masukan "b"



Gambar 6 Partisi ekuivalensi untuk masukan "c"

d. Partisi ekuivalensi untuk keluaran



Gambar 7. Partisi ekuivalensi untuk keluaran

b. Desain *Test Cases*

Dalam mendesain *test cases* di bawah ini digunakan pendekatan *one-to-one*

Tabel 1. Desain *Test cases* untuk partisi masukan tidak valid (1)

Test cases	**1**	**2**	**3**	**4**
Masukan "a"	129	129	60	90
Masukan "b"	-128	60	-128	40
Masukan "c"	-128	20	40	-128
Partisi yang dites	a>b && a>c	a>b && a>c	a>b && a>c	a>b && a>c
Keluaran yang diharapkan	Data tidak sesuai	Data tidak sesuai	Data tidak sesuai	Data tidak sesuai

Tabel 2. Desain *Test cases* untuk partisi masukan tidak valid (2)

Test cases	**5**	**6**	**7**	**8**
Masukan "a"	-128	-128	60	40
Masukan "b"	129	127	129	90
Masukan "c"	-128	20	40	-128
Partisi yang dites	b>a && b>c	b>a && b>c	b>a && b>c	b>a && b>c
Keluaran yang	Data tidak	Data tidak	Data tidak	Data tidak

diharapkan	sesuai	sesuai	sesuai	sesuai

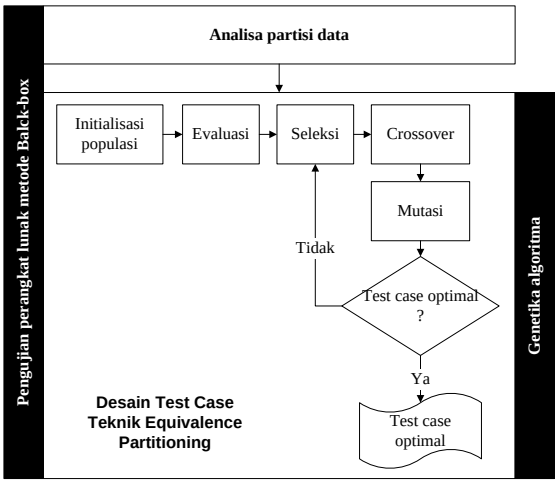
Tabel 3. Desain *Test cases* untuk partisi masukan tidak valid (3)

Test cases	**9**	**10**	**11**	**12**
Masukan "a"	-128	-128	60	40
Masukan "b"	-128	20	-128	90
Masukan "c"	129	60	90	129
Partisi yang dites	c>a && c>b	c>a && c>b	c>a && c>b	c>a && c>b
Keluaran yang diharapkan	Data tidak sesuai	Data tidak sesuai	Data tidak sesuai	Data tidak sesuai

Tabel 4. Desain *Test cases* untuk partisi masukan valid

Test cases	**13**	**14**	**15**
Masukan "a"	60	90	60
Masukan "b"	40	120	10
Masukan "c"	20	60	90
Partisi yang dites	a>b && a>c	b>a && b>c	c>a && c>b
Keluaran yang diharapkan	a	b	c

3. Metode yang Diusulkan



Gambar 3 Model algortima gentika dalam desain *test case* uji

Model yang menjadi usulan akan diterapkan pada perangkat lunak/ bahasa pemrograman matlab. Hasil luaran akan dilakukan pengukuran pencapaian tingkat optimasi dan efisiensinya.

3. Eksperimen Dan Pengujian Model/Metode

a. Reprensentasi

Setiap data dipresentasikan sebagai vektor biner yang berkaitan dengan input dari perangkat lunak yang diuji (Kusumadewi, Sri, 2003). Setiap bilangan atau nilai dari a, b atau c akan diwakili 8 bit sekaligus akan mengisi gen-gen, hal ini melihat bahwa panjang tipe data dari Byte adalah 8 bit. Sehingga panjang biner untuk mewakili satu set uji 8 X 9 atau 72. Hal tersebut terkait dengan nilai yang dicari a, b dan c dengan batasan tengah, batasan bawah dan batasan atas.

b. Penentuan parameter

Tabel 5. Penentuan parameter algortima gentika

Parameter	**Nilai**
Jumlah variabel (Nvar)	3
Jumlah bit untuk pengkodean satu variabel (Nbit)	8

Jumlah gen (JumGen)	Nvar*Nbit
Batas bawah interval (Rb)	-127
Batas atas interval (Ra)	128
Ukuran populasi (UkPop)	80
Jumlah generasi (MaxG)	100
Probabilitas *crossover* (Pc)	0,9
Probabilitas *mutation* (Pm)	0,01

c. Inisialisasi

Proses inisialisasi dilakukan dengan cara memberikan nilai awal gen-gen dengan nilai biner secara acak sesuai dengan batasan yang telah ditentukan. Sebagai contoh:

Chromosom[1] = Cakupan_uji[1], atau

Cakupan_uji[1] = [batas_tengah, batas_bawah, batas_atas], atau

Cakupan_uji[1] = {{00000101, 11110001, 00010100}, {00010000, 01111111, 01111111}, {11111111, 11110001, 00000110}}

d. Evaluasi individu

Sebelum dilakukan evaluasi terhadap *test case* atau individu dilakukan konversi dari biner kereal telebih dahulu sehingga perhitungan evaluasi/*fitness* dapat dilakukan. Dalam mengevaluasi individu didekatkan dengan metode *Equivalence Partitioning*. Sehingga untuk mengevaluasi objektif individu dapat ditentukan dengan perumusan (Agarwal, B. B.; Tayal, S.P.; Gupta, M., 2010):

$$B\_tengah = \left(1 - \left|Max(batas_{tengah})/255\right|\right) * \left(1 - \left|Min(batas_{tengah})/255\right|\right)$$

$$B\_bawah = \left(1 - \left|Max(batas_{bawah})/255\right|\right) * \left(1 - \left|Min(batas_{bawah})/255\right|\right)$$

$$B\_atas = \left(1 - \left|Max(batas_{atas})/255\right|\right) * \left(1 - \left|Min(batas_{atas})/255\right|\right)$$

Untuk mengkonversi fungsi objektif kedalam fungsi *fitness* dilakukan dengan perumusan:

$$Fitness = \frac{1}{(B_{tengah} * B_{bawah} * B_{atas}) + BilKecil}$$

BilKecil merupakan bilangan yang digunakan untuk menghindari pembagian dengan nol, misal 10^-1. Setelah nilai *fitness* diketahui berikutnya menskalakan nilai *fitness* kedalam ranking sehingga diperoleh nilai-nilai *fitness* baru yang berada dalam rentang [nilai fietness maximum, nilai fietness minimum].

Perumusan:

$$F(i) = F_{max} - (F_{max} - F_{min})(R(i) - 1/N - 1)$$

Keterangan:

Fmax: *fietness maximum*

Fmin: *fietness minimum*

R: populasi ke-i

N: ukuran populasi

e. Seleksi

Seleksi ini bertujuan untuk memberikan kesempatan reproduksi yang lebih besar bagi anggota populasi yang paling fit atau yang memiliki nilai *fitness* yang tinggi. Metode yang digunakan *roulette-wheel*, individu dipetakan dalam suatu segmen garis secara berurutan sedemikian hingga tiap-tiap segmen individu memiliki ukuran yang sama dengan ukuran *fietness*-nya. Sebuah bilangan random dibangkitkan dan individu yang memiliki segmen dalam kawasan bilangan random tersebut akan terseleksi. Proses ini diulang hingga diperoleh sejumlah individu yang diharapkan (Desiani, Anita; Arhami,

Muhammad, 2006) (Kusumadewi, Sri; Purnomo, Hari, 2010).

f. Crossover

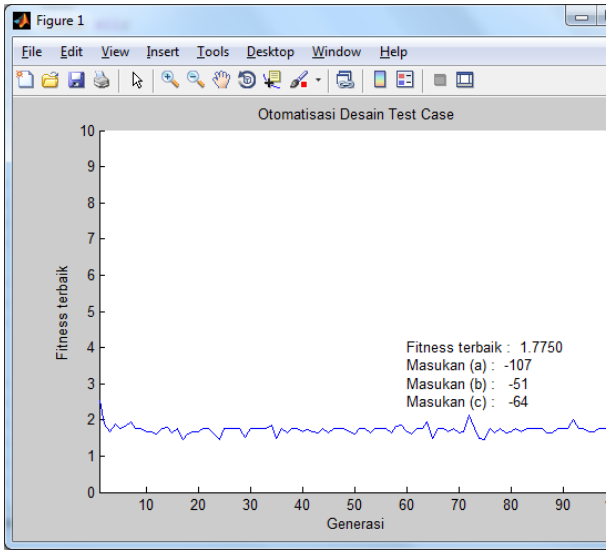
Metode *crossover* yang digunakan adalah satu titik dengan kromosom yang berbentuk string biner. Sebuah bilangan antara 1 sampai jumlah gen (JumGen) dibangkitkan secara acak untuk menentukan posisi persilangan. Kemudian ditukarkan bagian kanan titik potong dari kedua induk kromosom tersebut untuk menghasilkan kromosom anak (Kusumadewi, Sri, 2003).

g. Mutasi

Mutasi yang digunakan dengan kromosom biner. Mutasi terjadi secara acak pada setiap gen dalam kromosom. Kemudian mengganti bit 1 dengan 0 , atau mengganti bit 0 dengan 1. Pada mutasi ini sangat dimungkinkan munculnya kromosom baru yang semula belum muncul dalam populasi awal (Kusumadewi, Sri, 2003).

h. Pengujian model/metode

Dalam melakukan pengujian model yang diusulkan dilakukan running program sebanyak 12 kali.



Gambar 4 Runing program

Dari running program sampai ke 12 didapat desain *test case* sebagai berikut:

Tabel 6. Hasil running program desain *test case*

NO	Fitness terbaik	Variabel		
		A	B	C
a.	1.7750	-107	-51	-64
b.	2.5334	-119	114	72
c.	2.3422	-95	122	117
d.	2.3172	50	89	-125
e.	1.6763	-1	54	-94
f.	2.3068	-115	106	123
g.	2.5128	-12	-107	124
h.	2.6072	-126	112	-19
i.	1.9430	-70	-109	-91
j.	1.9606	13	49	-127
k.	1.9264	-4	105	-73
l.	2.3202	100	27	-116

4. Kesimpulan

Berdasarkan hasil yang telah dicapai terkait dengan Otomatisasi Desain *Test Case* Pengujian Perangkat Lunak Metode *Black-Box* Testing Dengan Teknik *Equivalence Partitioning* Menggunakan Algoritma Genetika. Maka dapat disimpulkan bahwa algoritma genetika dapat digunakan untuk mengotomatisasikan desain *test cases* pada pengujian perangkat lunak metode *black-box testing* dengan teknik *equivalence partitioning*.

5. Daftar Pustaka

Agarwal, B. B.; Tayal, S.P.; Gupta, M. (2010). *Software Engineering & Testing.* Sudbury, Massachusetts: Johanes and Bartlett Publishers.

Bau, Jason; Bursztein, Elie; Gupta, Divij; Mitchell, John. (2010, July). State of the Art Automated Black-BoxWeb Application Vulnerability. *Security and Privacy (SP)*, 332 - 345.

Bertolino, Antonia. (2007, June). Software Testing Research Achievements Challenges Dreams. *Future of Software Engineering*, 85-103.

Briand, Lionel C.; Labiche, Yvan; Bawar, Zaheer. (2008, August). Using Machine Learning to Refine Black-

Box Test Specifications and Test Suites. *Quality Software, 2008. QSIC '08*, 135 - 144.

Desiani, Anita; Arhami, Muhammad. (2006). *Konsep Kecerdasan Buatan.* (D. Harjono, Ed.) Yogyakarta, Indonesia: CV. Andi Offset.

Dick, S.; Kandel, A. (2005). *Computational Intelligence In Softeware Quality Assurance.* 5 Toh Tuck Link, Singapore: World Scientific Publishing.

Hendraputra, Ade; Pratondo, Agus; wijaya, Dedy Rahman; dkk;. (2009). *Jaminan Mutu Sistem Informasi.* (G. P. Kusuma, Ed.) Bandung, Indonesia: Politeknik Telkom.

Kusumadewi, Sri. (2003). *Artificial Intelligence.* Yogyakarta, Indonesia: Graha Ilmu.

Mitchell, Melanie. (1999). *An Introduction to Genetic Algorithm.* London, England: Massachusetts Institute of Technology.

Nedjah, Nadia; Abraham, Ajith; Mourelle, Luiza de Macedo. (2006). *Genetic Systems Programming.* (P. J. Kacprzyk, Ed.) Berlin Heidelberg, New York: Springer.

Rauf, Abdul; Anwa, Sajid; Jaffer, M. Arfan; Shahid, Arshad Ali. (2010, July). Automated GUI Test Coverage Analysis using GA. *Information Technology: New Generations (ITNG)*, 1057 - 1062.

Romeo. (2003). *Testing Dan Implementasi Sistem.* Surabaya: STIKOM.

Suyanto. (2005). *Algoritma Genetikan dalam Matlab.* Yogyakarto, Indonesia: Andi Offsite.