ISSN 2527-3671 (*Online*)
ISSN 2087-8893 (*Print*)

Teknologi: Jurnal Ilmiah Sistem Informasi 10 (1) Januari 2020 1-9

Tersedia *online* di www.journal.unipdu.ac.id
Unipdu
Terakreditasi Sinta S5
Halaman jurnal di www.journal.unipdu.ac.id/index.php/teknologi

Pengembangan sistem otomatisasi pembangkitan kasus uji dengan algoritma genetika dan *test case generation method*

Development of automation systems for generating test cases with genetic algorithms and test case generation methods

Moh Arsyad Mubarak Setyawan [a], Fajar Pradana [b], Bayu Priyambadha [c]

[a,b,c] *Teknik Informatika, Universitas Brawijaya, Malang, Indonesia*

email: [a] arzad.mubarak@gmail.com, [b] fajar.p@ub.ac.id, [c] bayu_priyambadha@ub.ac.id

INFO ARTIKEL

Sejarah artikel:
Menerima 2 November 2016
Revisi 6 Februari 2018
Diterima 7 Februari 2018
Online 5 April 2020

Kata kunci:
algoritma genetika
jalur independen
kasus uji
test case generation

Keywords:
genetic algorithm
independenth path
test case
test case generation

Style APA dalam menyitasi artikel ini:
Setyawan, M. A., Pradana, F., & Priyambadha, B. (2020). Pengembangan sistem otomatisasi pembangkitan kasus uji dengan algoritma genetika dan method test case generation. *TEKNOLOGI: Jurnal Ilmiah Sistem Informasi*, 10(1), 1-9.

ABSTRAK

Pengujian perangkat lunak merupakan salah satu bagian penting dari pembuatan perangkat lunak. Pada pengujian perangkat lunak terdapat pengujian unit. Pengujian unit merupakan proses pengujian komponen yang berfokus untuk memverifikasi unit terkecil pada perancangan perangkat lunak. Pada tahap pengujian unit terdapat proses pembangkitan kasus uji. Selama ini, pembangkitan kasus uji dari suatu kode program dilakukan secara manual sehingga membutuhkan waktu yang lama. Hal ini dikarenakan banyaknya kemungkinan jalur pada kode sumber yang akan diuji. Dalam penelitian ini dibangun suatu sistem otomatis untuk membangkitkan kasus uji. Alur kerja sistem dimulai dari analisa kode sumber dengan *Spoon Library*, selanjutnya dibentuk CFG (*Control Flow Graph*) dan DDG (*Dynamic Directed Graph*). Dari DDG tersebut akan dibangkitkan jalur layak yang terdapat pada DDG, dengan menggunakan algoritma genetika diharapkan dapat mengoptimalkan penentuan jalur independen. Dari masing-masing jalur independen akan dibangkitkan kasus ujinya dengan metode *test case generation*. Pengujian akurasi sistem pada sistem otomatisasi pembangkit kasus uji dengan jumlah populasi 5, 10 dan 15 serta jumlah maksimum generasi 50, 100, 200 dan 250 dihasilkan jumlah populasi paling optimal yaitu 10 dan maksimum generasi optimal yaitu 200 dengan akurasi 93,33%. Pada jumlah populasi dan maksimum generasi sesudahnya tidak terjadi peningkatan akurasi yang signifikan. Tiap peningkatan jumlah populasi dan maksimum generasi dapat meningkatkan akurasi sistem.

ABSTRACT

Software testing is one of the most important part of making software. On the software testing there are unit testing. Unit Testing is a process for verifying component, focusing on the smallest unit of software design. In the unit testing phase contained test case generation process. During this time, the generation of test cases of a program code is done manually. In this study, constructed an automated system to generate test cases. The workflow system starts from the analysis of the source code with the library spoon and then create CFG (Control Flow Graph) and DDG (Dynamic Directed graph). From the DDG will be raised feasible path using a genetic algorithm. Furthermore, from fea-sible path sought independenth path which is a path base d on the level of uniqueness of the path to the other path. From each independenth path raised the test case with a test case generation method. Testing accuracy of the system on the automation system generating test cases with populations of 5,10 and 15 as well as the maximum number of generations 50, 100, 200 and 250 produced the most optimal population number is 15 and the most optimal maximum generation is 200 with accuracy 93.33%. Each increase in the number of population and maximum generation can improve the accuracy of the system. Level

ISSN 2527-3671 (*Online*)
M. A. M. Setyawan dkk./Teknologi 10 (1) Januari 2020 1-9 ISSN 2087-8893 (*Print*)

accuracy with population number over 10 and maximum generation over 200 has no increace accuracy significant.

1. Pendahuluan

Pengujian perangkat lunak merupakan aspek yang sangat diperlukan, karena secara langsung berpengaruh pada kualitas perangkat lunak pada pelaksanaan proses pengujian yang sistematik memerlukan *effort* dalam menguji *software* (Chaudhary & Yadav, 2012). Proses pengujian sendiri terdiri dari beberapa tahap, diantaranya modul pengujian, pengujian integrasi, pengujian sistem, dan pengujian instalasi. Semua aktivitas pengujian dari modul yang berbeda tersebut harus diselesaikan dalam waktu yang terbatas dan aktivitas ini normalnya mengkonsumsi sekitar 40-50% dari total jumlah sumber daya pengembang (Htoon & Thein, 2005).

Selain itu terdapat *Test Case Point* (TCP) analisis yang mana merupakan pendekatan untuk melakukan estimasi akurat pada pengujian fungsional projek. Analisis TCP menghasilkan *test effort* untuk beberapa aktivitas pengujian yang terpisah. *Test effort* merupakan faktor kunci dalam melakukan *testing*, karena hal ini yang menentukan kompleksitas dari keseluruhan siklus *testing* mulai dari usaha menerjemahkan pembuatan *test* sampai usaha pengeksekusian *test*. Analisis TCP tersebut terdiri dari empat modul yang berbeda yaitu: TCP, *automated script generation*, *manual test execution*, dan *automated test execution* (Chaudhary & Yadav, 2012).

Pada tahap *test case generation* terjadi pembuatan *test case*. Estimasi jumlah *test case* yang dibuat ditentukan oleh estimasi *function point* (Capers, 1996). Selanjutnya usaha pembuatan *test case* dalam *person-hours* dihitung. Dalam hal ini, waktu dalam pembuatan *test case* dapat dipersingkat jika dilakukan secara otomatis.

Pada proses pembangkitan *test case* yang konvensional menurut Watson dan McCabe (1996) memiliki empat langkah, yaitu menghitung *graph* program, menghitung *cyclomatic complexity*, memilih himpunan basis *path*, dan membangkitkan *test case* dari tiap *path*. Saat membangkitkan himpunan basis *path* secara otomatis peneliti menggunakan proses algoritma genetika yang diusulkan oleh Ahmed S. Ghiduk (Ghiduk, 2014). Algoritma genetika yang diusulkan terdiri dari proses *encoding*, *initial population*, *evaluation function*, *selection*, *reproduction*, dan *elitist*. Berdasarkan dua algoritma tersebut, peneliti membangun suatu sistem pembangkit *test case* dengan algoritma genetika dan *test case generation* pada kode program Java. Konsep dari sistem yang dibangun ialah menerjemahkan suatu kode program Java dari suatu Java *file* ke dalam jalur *independent path.*

Selanjutnya, jalur tersebut dianalisa untuk setiap *test case* yang mendukung masing-masing jalur tersebut. Dengan dibuatnya sistem otomatis pembangkit kasus uji dengan algoritma genetika dan *test case generation* pada kode diharapkan dapat mempercepat waktu, menghemat tenaga dan biaya dalam tahap *test case generation* pada *software testing*.

2. *State of the Art*

2.1. *Basis path testing*

Basis path testing merupakan salah satu kriteria struktural *testing* yang kuat. *Basis path testing* membutuhkan jumlah dari *test path* sama terhadap *cyclomatic complexity* program, di mana tiap *path* merupakan *independent path* dan semua *edges* dalam *Control-Flow Graph* (CFG) dicakup seluruh *path* dalam himpunan *basis path* (Ghiduk, 2014). Metode yang ditemukan Watson dan McCabe (1996) dalam menangani *basis path testing* memiliki empat langkah, yaitu 1) Menghitung *graph* program, 2) Menghitung *cyclomatic complexity*, memilih himpunan *basis path*, *generating test cases* dari tiap *path*.

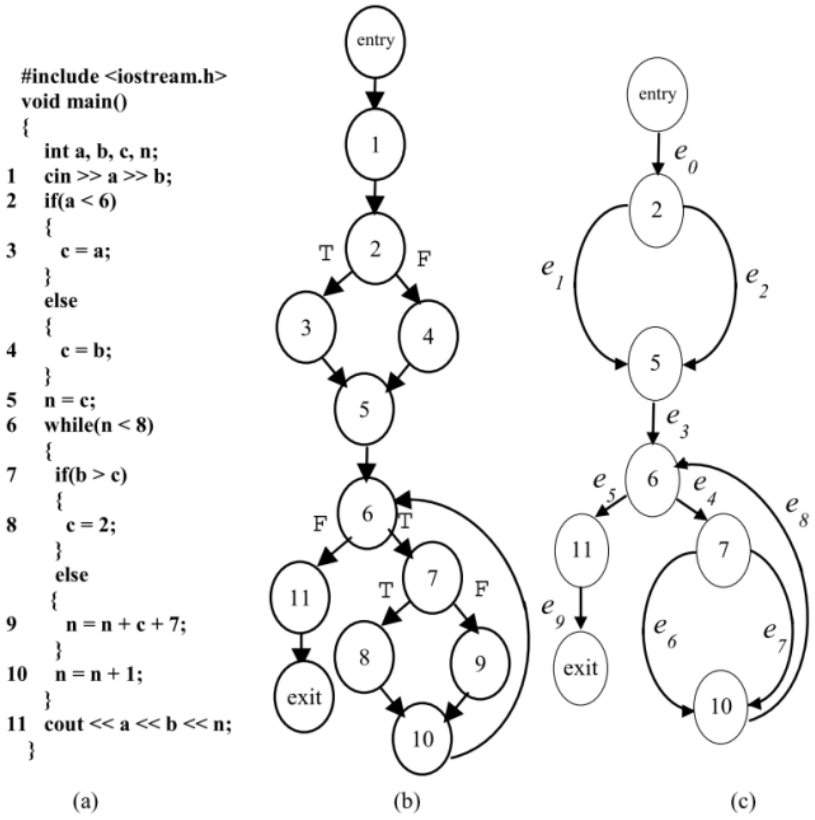
Independent path merupakan *path* dalam program, di mana sekurangnya satu *edge* dari *path* tersebut tidak pernah tampak *path* yang lain dalam CFG *basis set of paths and basis path*. Sebuah himpunan *basis path* merupakan himpunan dari *path*. Setiap *path* harus memenuhi tiga kondisi berikut: Setiap *path* harus sebuah *independent path*, semua *edges* dalam CFG harus dicakup semua *path* dalam himpunan *basis*. Setiap *path* tidak mengandung himpunan *basis path* yang dapat dibentuk dari operasi linier di antara *path* dalam himpunan tersebut.

2.2. Algoritma genetika untuk otomatisasi generasi dari *basis test path*

Genetic Algorithm (GA) yang diusulkan Ghiduk (2014) untuk menghasilkan *basis test path* dari *software*

ISSN 2527-3671 (*Online*)
M. A. M. Setyawan dkk./Teknologi 10 (1) Januari 2020 1-9 ISSN 2087-8893 (*Print*)

yang diuji secara otomatis, di mana menggunakan fungsi *fitness* baru untuk mengevaluasi *basis path* yang dihasilkan. Contoh kasus program dengan CFG dan DDG yang diambil terlihat pada Gambar 1.



Gambar 1. Algoritma Genetik

Adapun penjelasan dari Gambar 1 adalah sebagai berikut:

1) Lingkup pencarian algoritma
Lingkup pencarian algoritma dirumuskan seperti pada Persamaan 1,
$$\boldsymbol{D} = \{\forall \boldsymbol{e} | \boldsymbol{e} \in \boldsymbol{DDG\ dan\ e\ dicapai\ dari\ entry\ dan\ mencapai\ exit}\} \quad (1)$$
di mana *D* merupakan lingkup pencarian algoritma genetika dan *e* dan *edge* dari *dd-graph*. Sebagai contoh pada Gambar 1 himpunan dari *edges* $\boldsymbol{D} = \{\boldsymbol{e_0}, \boldsymbol{e_1}, \boldsymbol{e_2}, \boldsymbol{e_3}, \boldsymbol{e_4}, \boldsymbol{e_5}, \boldsymbol{e_6}, \boldsymbol{e_7}, \boldsymbol{e_8}, \boldsymbol{e_9}\}$.

2) *Encoding*
Encoding merupakan proses merepresentasikan gen individu, dengan Persamaan 2,
$$\boldsymbol{M}: \boldsymbol{i} \in \boldsymbol{dd-graph} \quad (2)$$
di mana *M* merupakan *mapping* dari kromosom dengan *i* sebagai *index* kromosom dan $\boldsymbol{e_i}$ merupakan *index* dari *edges* pada $\boldsymbol{dd-graph}$. Sebagai contoh pada Gambar 1 contoh kromosom: $(\boldsymbol{0}, \boldsymbol{1}, \boldsymbol{3}, \boldsymbol{5}, \boldsymbol{9})$ dengan menggunakan *M*, kromosom dipetakan dalam *path* $\boldsymbol{p} = \{\boldsymbol{e_0}, \boldsymbol{e_1}, \boldsymbol{e_3}, \boldsymbol{e_5}, \boldsymbol{e_9}\}$.

3) *Initial population*
Setiap *chromosome* direpresentasikan sebagai *vector integer*. Kami secara acak menghasilkan *population size* (PS) *integer vector* (*chromosome*) dengan panjang 2 untuk merepresentasikan populasi awal, di mana PS sebagai panjang ukuran populasi.

4) *Evaluation function*
Dalam menghitung nilai *fitness* $\boldsymbol{ft(vi)}$ untuk setiap *chromosome* $\boldsymbol{vi}$ $(\boldsymbol{i} = \boldsymbol{1}, \ldots, \boldsymbol{PS})$ dihitung sebagai berdasarkan Persamaan 3,
$$ft(vi)^n = \sum_{j=1}^{d(vi)} w(ej) \quad (3)$$
di mana, $d(vi)$ adalah jumlah *edges* berdekatan dalam kromosom $vi, w(e\,j)$ adalah berat (*probability*) dari *edge* $e\,j$ dalam kromosom $vi, w(e\,j) = L(1\,vi)$, di mana $L(vi)$ panjang dari kromosom vi. Fungsi *fitness* dapat ditulis dengan jumlah *edges* yang berdekatan dibagi dengan total jumlah *edges* dalam kromosom yang sama.

5) *Selection*

M. A. M. Setyawan dkk./Teknologi 10 (1) Januari 2020 1-9
ISSN 2527-3671 (*Online*)
ISSN 2087-8893 (*Print*)

Proses seleksi yang digunakan dalam GA yang diusulkan menggunakan *roulette wheel*. Proses seleksi didasarkan pada pemutaran *roulete wheel* sebanyak populasi. Setiap kali kita memilih satu *single* kromosom untuk populasi baru.

6) *Reproduction*
Proses reproduksi dibagi ke dalam tiga proses yaitu *crossover*, *mutation*, dan *breeding*. Penjelasan dari tiap proses tersebut dijelaskan sebagai berikut:
   a. *Crossover*: GA yang diusulkan menggunakan *uniform crossover* untuk menukar informasi pada posisi *random* dua kromosom yang terpilih untuk menghasilkan dua kromosom baru. Setiap pasangan kromosom menghasilkan sebuah bilangan *random integer* pos dari rentang [$\mathbf{2}\ldots\boldsymbol{L}-\mathbf{1}$] ($L$ adalah jumlah *edges* dalam kromosom).
   b. *Mutation*: Prosedure pada proses mutasi GA yang diusulkan sebagai berikut:
      1. Hasilkan sebuah bilangan *random r* dari rentang [$\mathbf{0}\ldots\mathbf{1}$];
      2. Jika $\boldsymbol{r} < \boldsymbol{MP}$ maka mutasikan *cell* dengan mengganti *edge* dengan *edges* lain yang merupakan *siblings* (*edges* dengan *parent* sama disebut *siblings*)
   c. *Breeding*: Memungkinkan interaksi terbatas antara sub-populasi melalui prosedur 'pembiakan silang' di mana anggota sub-populasi yang dipilih berbaur untuk membentuk populasi gabungan untuk optimasi lebih lanjut (Pham & Karaboga, 1998).

7) *Elitist*
Fungsi *elitist* meningkatkan populasi sekarang dengan menyimpan satu *copy* anggota terbaik dari populasi sebelumnya.

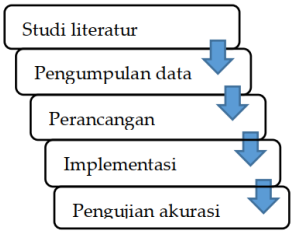
8) *The stop condition*
Terdapat dua kasus kondisi berhenti. Pertama, *test path* yang dihasilkan telah memenuhi kondisi himpunan *basis path*. Kedua, jumlah generasi mencapai jumlah maksimum generasi.

2.3. *Test case generation methods*

Algoritma ini Wijayasiriwardhane, Wijayarathna, & Karunarathna (2011) secara simbolis mengeksekusi tiap eksekusi *path* yang dilewati program berdasarkan pada *elements* dari sebuah *basis path* yang terdapat pada *flow graph* dan membangkitkan *test case* yang terdapat pada *basis path testing*. Algoritma ini mengambil *independent path* secara linier dari *flow graph* dan melintasi jalur dari node sumber ke node akhir dengan mengunjungi semua node pada jalur. Ketika node dikunjungi, algoritma ini mengeksekusi *statement* yang berhubungan dalam program secara simbolis. Algoritma ini memperlakukan input variabel dan parameter pada program sebagai input *test case*. Hal ini, menghasilkan *expected output* dan ekspresi dari setiap *decision* yang dibuat selama mengekseskusi jalur yang berhubungan dengan *input test case*.

2.4. *Spoon*

Spoon merupakan sebuah *library* yang dapat digunakan untuk menganalisa dan merubah kode program dengan bahasa Java. *Spoon* memungkinkan pengembang untuk menulis dengan cakupan besar dalam menganalisa domain yang spesifik dan merubahnya dengan mudah sesuai kebutuhan (Pawlak, Monperrus, Petitprez, Noguera, & Seinturier, 2016).



Gambar 2. Metode penelitian

3. Metode Penelitian

Metode yang dilakukan dalam penelitian ini dikerjakan dengan beberapa tahapan, antara lain pengumpulan data, perancangan algoritma, implementasi sistem, pengujian akurasi. Gambar 2 menggambarkan urutan metode dalam penelitian ini.

3.1. Studi literatur

ISSN 2527-3671 (*Online*)
M. A. M. Setyawan dkk./Teknologi 10 (1) Januari 2020 1-9 ISSN 2087-8893 (*Print*)

Studi literatur merupakan metode yang digunakan untuk mencari dasar konsep dan kajian yang dapat digunakan dalam penelitian ini. Literatur tersebut dapat diperoleh dari buku, jurnal, buku, penelitian sebelumnya dan dokumentasi *project*. Bagian studi literatur ini mencakup teori diantaranya adalah sebagai berikut:

a. *Basis path testing*
b. Algoritma genetika untuk otomatisasi generasi dari *basis path*
c. *Test case generation*
d. *Spoon*

3.2. Pengumpulan data

Pada penelitian ini pengumpulan data didapatkan dari *file* kode sumber berekstensi Java yang diperoleh dari website freesourcecode.net. Dari *file* tersebut hanya diambil beberapa metode saja. Metode tersebut diambil sebagian untuk melakukan uji coba kakas bantu pembangkitan jalur independen. Tabel 1 menunjukan metode yang digunakan pada penelitian ini.

Tabel 1. Metode uji coba

Nama Kelas	Nama Metode	Kode Program	Jumlah Kasus Uji Coba
InsertionSort	Insert	M#1	4
BinaryHeap	getFirst	M#2	2
BinaryHeap	Remove	M#3	2
BinaryHeap	Clear	M#4	2
BinaryHeap	rebuildHeap	M#5	5
Prim	Weight	M#6	3
Prim	findNode	M#7	3
Prim	draw_arrow	M#8	2
Prim	remove_pre_s	M#9	3
Prim	step3	M#10	4
Prim	paintEdge	M#11	11
Prim	paintNode	M#12	8
Prim	init_sub	M#13	4
Prim	mousePressed	M#14	4
Kruskal	isPresent	M#15	4

3.3. Perancangan

Perancangan algoritma yang digunakan pada penelitian ini menggunakan GA (Ghiduk, 2014). Parameter-parameter pada algoritma tersebut antara lain, PopSize merupakan ukuran populasi dari GA, maksimum generasi merupakan jumlah maksimal generasi yang dihasilkan, *Probability Crossover* merupakan peluang dari terjadinya proses *crossover* pada suatu populasi, *Probability Mutation* merupakan peluang dari terjadinya proses mutasi pada suatu populasi. *Edge* merupakan gen dari suatu kromosom yang terdiri dari node awal dan berakhir di *node* akhir dari suatu lintasan pada DDG.

Panjang kromosom dari suatu individu bervariasi mulai dari 2 hingga maksimum generasi. Hal dikarenakan pada tiap generasi panjang kromosom bertambah satu. GA yang diusulkan Ghiduk (2014) untuk menghasilkan *basis test path* dari perangkat lunak yang diuji secara otomatis, di mana menggunakan fungsi *fitness* baru untuk mengevaluasi *basis path* yang dihasilkan. Proses awal yang dilakukan dalam Algoritma Genetika yaitu dilakukan inisialisasi populasi. Setiap kromosom direpresentasikan sebagai *integer vector*.

Secara acak menghasilkan PS *integer vector* (kromosom) dengan panjang 2 untuk merepresentasikan populasi awal, di mana PS sebagai panjang ukuran populasi. Contoh dari inisialisasi populasi dijelaskan pada Tabel 2.

Dari setiap kromosom dilakukan perhitungan nilai *fitness* $ft(vi)$ untuk setiap kromosom vi $(i = 1, \ldots, PS)$ dihitung menggunakan Persamaan 4,

$$ft(vi)^n = \sum_{j=1}^{d(vi)} w(ej) \quad (4)$$

di mana, $d(vi)$ adalah jumlah *edges* berdekatan dalam kromosom *vi*, *w(e j)* adalah berat (*probability*) dari *edge e j* dalam kromosom *vi*, *w(e j) = L(1 vi)* , di mana *L(vi)* panjang dari kromosom *vi*. Fungsi *fitness* dapat ditulis dengan jumlah *edges* yang berdekatan dibagi dengan total jumlah *edges* dalam kromosom yang

ISSN 2527-3671 (*Online*)
M. A. M. Setyawan dkk./Teknologi 10 (1) Januari 2020 1-9 ISSN 2087-8893 (*Print*)

sama. Proses awal dari algoritma genetika ini dengan melakukan proses *crossover*. Dalam proses *crossover*, untuk menentukan pasangan dilakukan proses seleksi menggunakan *roulette wheel*. Proses seleksi didasarkan pada pemutaran *roulette wheel* sebanyak populasi. Contoh dari proses ini diperlihatkan pada Tabel 3.

Tabel 2. Inisialisasi populasi (Ghiduk, 2014)

Kromosom	Fitness
$e_0 e_9$	0,0
$e_0 e_9$	0,0
$e_0 e_9$	0,0
$e_0 e_9$	0,0

Tabel 3. Contoh proses seleksi

C #	Jalur	RF	CF	r
C1	e_0, e_1, e_3, e_5, e_9	0,25	0,25	0,70
C2	e_0, e_2, e_3, e_4, e_9	0,25	0,50	0,20
C3	e_0, e_2, e_3, e_5, e_9	0,25	0,75	0,80
C4	e_0, e_1, e_3, e_4, e_9	0,25	1,00	0,15

Pada Tabel 3 kolom C# merupakan kromosom kesekian dari suatu populasi. Kolom RF merupakan nilai *relative fitness* dari masing-masing kromosom yang ada. Nilai ini didapatkan dari 1 dibagi dengan ukuran populasi. Kolom CF merupakan nilai *cumulative fitness* yang dihitung dengan nilai *relative fitness* dari suatu kromosom ditambah dengan nilai *cumulative fitness* sebelumnya. Kolom *r* merupakan nilai random dari 0 sampai 1 sebagai acuan memilih pasangan dari suatu kromosom. Kolom *P* merupakan pasangan yang didapatkan.

Tahap selanjutnya yaitu proses reproduksi dibagi ke dalam tiga proses yaitu *crossover*, *mutation*, dan *breeding*. Penjelasan dari tiap proses tersebut dijelaskan sebagai berikut:

a. *Crossover*: GA yang diusulkan menggunakan *uniform crossover* untuk menukar informasi pada posisi *random* dua kromosom yang terpilih untuk menghasilkan dua kromosom baru. Setiap pasangan kromosom menghasilkan sebuah bilangan *random integer* pos dari rentang $[2 \ldots . L - 1]$ (L adalah jumlah *edges* dalam kromosom).
   Contoh dari proses *crossover* sebagai berikut:
   Individu e_0, e_2, e_3, e_4, e_9 dengan e_0, e_1, e_3, e_5, e_9 dilakukan *crossover* dengan titik potong posisi 3 menghasilkan *offspring* e_0, e_2, e_3, e_5, e_9 dan e_0, e_1, e_3, e_4, e_9.
b. *Mutation*: prosedur pada proses mutasi GA yang diusulkan sebagai berikut. Hasilkan sebuah bilangan *random r* dari rentang [0...1]; Jika $r < MP$ maka mutasikan *cell* dengan mengganti *edge* dengan edges lain yang merupakan siblings (edges dengan parent sama disebut siblings).
   Contoh dari proses *mutation* diperlihatkan pada Tabel 4.

Tabel 4. Contoh proses seleksi

Kromosom	*R*	Kromosom Baru
e_0, e_1, e_3, e_4, e_9	0,5; 0,1; 0,1; 0,2; 0,1	0,25

Pada Tabel 4, C1 merupakan kromosom ke-1 dari populasi. *R* merupakan nilai *random* masing-masing gen. Pada C1 diketahui bahwa nilai *R* terkecil dan paling awal dijumpai dari gen yang terdapat pada C1 adalah e_1 dengan nilai *R* 0,1. Oleh karena itu, e_1 digantikan dengan *sibling* yaitu e_2. Sehingga C1 baru yang dihasilkan adalah e_0, e_2, e_3, e_4, e_9. Proses ini akan diulangi sebanyak $PM \times popSize \times lengthCell = 0{,}15 \times 4 \times 5 = 10$. *Breeding* adalah operator *breeding* berjalan dengan cara berikut:

1. Hasilkan sebuah bilangan *random* pos dari rentang $[2 \ldots . L - 1]$, L merupakan panjang kromosom. Bilangan pos mengindikasikan titik *breeding*.

ISSN 2527-3671 (*Online*)
M. A. M. Setyawan dkk./Teknologi 10 (1) Januari 2020 1-9 ISSN 2087-8893 (*Print*)

2. Identifikasikan *edge* pada posisi pos dan secara *random* pilih suksesor *edges* tersebut. Lalu, sisipkan suksesor *edges* pada posisi pos + 1 dan tambahkan panjang kromosom satu.

Contoh dari proses *breeding* diperlihatkan pada Tabel 5.

Tabel 5. Contoh proses *breeding*

Populasi Sekarang	Suksesor	Populasi Baru
$e_0, e_2, e_3, \mathbf{e_5}, e_9$	e_9	$e_0, e_2, e_3, e_4, e_9, e_9$
$e_0, e_2, e_3, \mathbf{e_4}, e_9$	e_6, e_7	$e_0, e_1, e_3, e_4, e_6, e_9$
$e_0, e_2, e_3, \mathbf{e_5}, e_9$	e_9	$e_0, e_2, e_3, e_5, e_9, e_9$
$e_0, e_1, e_3, \mathbf{e_5}, e_9$	e_9	$e_0, e_1, e_3, e_5, e_9, e_9$

Setelah proses reproduksi selesai, tahap selanjutnya dilakukan proses *elitist*. Fungsi *elitist* meningkatkan populasi sekarang dengan menyimpan satu *copy* anggota terbaik dari populasi sebelumnya. Jika anggota terbaik dari populasi sebelumnya lebih baik dari anggota terbaik dari populasi sekarang, maka anggota terburuk dari populasi sekarang digantikan dengan anggota terbaik dari populasi sebelumnya. Setelah semua proses dalam algoritma genetika dilakukan, akan dilakukan pengecekan kondisi berhenti. Terdapat dua kasus kondisi berhenti. Pertama, *test path* yang dihasilkan telah memenuhi kondisi himpunan *basis path*. Kedua, jumlah generasi mencapai jumlah maksimum generasi.

3.4. Implementasi

Implementasi sistem merupakan tahapan membangun sistem otomatisasi pembangkitan kasus uji dengan menggunakan bahasa pemrograman Java yang didasarkan pada perancangan yang telah dibuat. Tahapan-tahapan yang ada dalam implementasi antara lain:

1. Membuat antarmuka sistem menggunakan tampilan dengan *library* java.awt.swing.
2. Implementasi parsing kode sumber menggunakan *library Spoon*. Implementasi ini digunakan untuk membaca masukan kode sumber dan membentuk CFG dan DDG dari kode sumber tersebut.
3. Implementasi algoritma genetika berdasarkan perancangan algoritma genetika yang dibuat implementasi metode *test case generation* berdasarkan *paper* Wijayasiriwardhane, Wijayarathna, dan Karunarathna (2011).

3.5. Pengujian akurasi

Pengujian akurasi sistem dilakukan dengan masukan jumlah populasi 5, 10 dan 15 serta jumlah maksimum generasi 50, 100, 200 dan 250. Tahapan pengujian yang dilakukan terdiri dari beberapa tahapan yaitu:

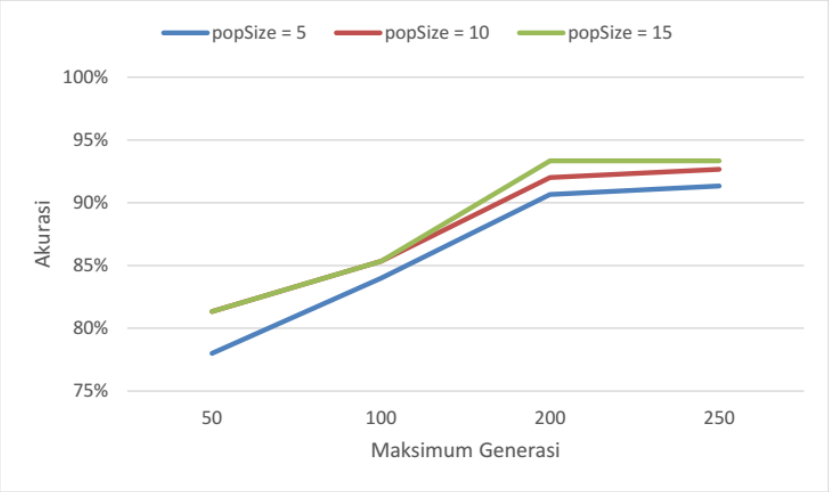
1. Lakukan proses pembangkitan kasus uji dari *file* kode sumber menggunakan sistem dan secara manual.
2. Lakukan proses pembangkitan pada suatu metode sebanyak 10 kali dengan jumlah populasi dan maksimum generasi tertentu.
3. Hitung jumlah seluruh kasus uji yang benar untuk semua metode.
4. Hitung akurasi berdasarkan jumlah kasus uji yang benar dibagi jumlah keseluruhan proses pembangkitan.

4. Hasil dan Pembahasan

Hasil dan pembahasan merupakan tahap pengujian hasil dari perancangan dan implementasi sistem otomatisasi pembangkitan kasus uji. Berikut ini merupakan grafik hasil pengujian akurasi sistem yang telah dilakukan, ditunjukkan pada Gambar 3.

Pada Gambar 3 menunjukkan bahwa peningkatan jumlah populasi dan maksimum generasi dapat meningkatkan akurasi sistem. Pengujian akurasi sistem pada sistem otomatisasi pembangkit kasus uji dengan jumlah populasi 5, 10 dan 15 serta jumlah maksimum generasi 50, 100, 200 dan 250 dihasilkan jumlah populasi paling optimal yaitu 10 dan maksimum generasi optimal yaitu 200 dengan akurasi 93,33%. Pada jumlah populasi dan maksimum generasi sesudahnya tidak terjadi peningkatan akurasi yang signifikan. Tiap peningkatan jumlah populasi dan maksimum generasi dapat meningkatkan akurasi sistem. Peningkatan jumlah populasi yang dapat meningkatkan akurasi sistem dikarenakan semakin banyak populasi maka semakin banyak kombinasi yang dilakukan sehingga lebih banyak ditemukan *feasible path*.

M. A. M. Setyawan dkk./Teknologi 10 (1) Januari 2020 1-9
ISSN 2527-3671 (*Online*)
ISSN 2087-8893 (*Print*)



Gambar 3. Grafik uji akurasi

Sedangkan, peningkatan maksimum generasi dapat meningkatkan jumlah kombinasi kromosom dan panjang kromosom. Dari penjelasan mengenai peningkatan jumlah populasi dan maksimum generasi dapat diketahui bahwa, penyebab dari kurangnya akurasi adalah kurangnya kombinasi kromosom yang dilakukan dan kurangnya panjang kromosom untuk suatu path yang memiliki panjang melebihi jumlah maksimum generasi.

Dengan nilai akurasi 93,33% dapat dikatakan bahwa sistem otomatisasi yang dikembangkan sesuai dengan yang diharapkan. Sistem ini dapat menggantikan proses perhitungan secara manual, sehingga dapat mempersingkat waktu dan biaya dalam proses pengujian perangkat lunak khususnya pada pengujian unit method jika dilakukan dengan manual.

5. Kesimpulan

Berdasarkan perancangan, implementasi dan hasil pengujian dari sistem, maka didapat kesimpulan, di antaranya adalah 1) Dalam mendukung proses algoritma genetika dibangun dua kelas yaitu *Edge* sebagai gen dan *Path* sebagai kromosom. Sistem otomatisasi pembangkit kasus uji memiliki tiga fitur yaitu memasukkan *file* kode sumber, membangkitkan kasus uji dan menyimpan hasil pembangkitan kasus uji, 2) Pengujian akurasi sistem pada sistem otomatisasi pembangkit kasus uji dengan jumlah populasi 5, 10 dan 15 serta jumlah maksimum generasi 50, 100, 200 dan 250 dihasilkan jumlah populasi paling optimal yaitu 10 dan maksimum generasi optimal yaitu 200 karena jumlah populasi dan maksimum generasi sesudahnya tidak terjadi peningkatan akurasi yang signifikan. 3) Pengujian akurasi sistem adalah 93,33 %, sehingga bisa dikatakan memiliki tingkat keakuratan tinggi. Sehingga dengan menggunakan sistem ini pengguna dapat mempersingkat waktu dan biaya jika dibandingkan dengan manual. 4) Algoritma Genetika memiliki karakteristik yaitu semakin banyak iterasi hasilnya akan semakin akurat. Namun apabila iterasi yang sama diberlakukan pada semua tingkat kompleksitas kode terlalu berlebihan. Perlu dilakukan penelitian lebih lanjut mengenai iterasi yang optimal pada penerapan algoritma genetika. Sehingga iterasi GA disesuaikan dengan tingkat kompleksitas kode.

7. Referensi

Capers, J. (1996). *Applied software measurement* (Second Edition ed.). McGraw-Hill.

Chaudhary, P., & Yadav, C. S. (2012). An Approach for Calculating the Effort Needed on Testing Projects. *International Journal of Advanced Research in Computer Engineering & Technology, 1*(1), 035-040.

Ghiduk, A. S. (2014). Automatic generation of basis test paths using variable length genetic algorithm. *Information Processing Letters, 114*(6), 304-316. doi:https://doi.org/10.1016/j.ipl.2014.01.009

Htoon, C., & Thein, N. L. (2005). Model-based Testing Considering Cost, Reliability and Software Quality. *6th Asia-Pacific Symposium on Information and Telecommunication Technologies.* Yangon, Myanmar: IEEE. doi:https://doi.org/10.1109/APSITT.2005.203649

Pawlak, R., Monperrus, M., Petitprez, N., Noguera, C., & Seinturier, L. (2016). SPOON: A library for implementing analyses and transformations of Java source code. *Software: Practise and Experience, 46*(9), 1155-1179. doi:https://doi.org/10.1002/spe.2346

ISSN 2527-3671 (*Online*)
M. A. M. Setyawan dkk./Teknologi 10 (1) Januari 2020 1-9 ISSN 2087-8893 (*Print*)

Pham, D. T., & Karaboga, D. (1998). Cross breeding in genetic optimisation and its application to fuzzy logic controller design. *Artificial Intelligence in Engineering, 12*(1–2), 15-20. doi:https://doi.org/10.1016/S0954-1810(96)00034-9

Watson, A. H., & McCabe, T. J. (1996). *Structured Testing: A Testing Methodology Using the Cyclomatic Complexity Metric.* Gaithersburg, Maryland: National Institute of Standards and Technology (NIST).

Wijayasiriwardhane, T. K., Wijayarathna, P. G., & Karunarathna, D. D. (2011). An automated tool to generate test cases for performing basis path testing. *International Conference on Advances in ICT for Emerging Regions (ICTer).* Colombo, Sri Lanka: IEEE. doi:https://doi.org/10.1109/ICTer.2011.6075032