ALGORITMA *SIMULATED ANNEALING* UNTUK PEMBANGKITAN *DATA TEST* SECARA OTOMATIS PADA PENGUJIAN PERANGKAT LUNAK

SKRIPSI

Oleh:
Ainun Najib
NIM. 15650053

JURUSAN TEKNIK INFORMATIKA
FAKULTAS SAINS DAN TEKNOLOGI
UNIVERSITAS ISLAM NEGERIMAULANA MALIK IBRAHIM
MALANG
2020

CENTRAL LIBRARY OF MAULANA MALIK IBRAHIM STATE ISLAMIC UNIVERSITY OF MALANG

ALGORITMA *SIMULATED ANNEALING* UNTUK PEMBANGKITAN *DATA TEST* SECARA OTOMATIS PADA PENGUJIAN PERANGKAT LUNAK

SKRIPSI

Diajukan kepada:
Universitas Islam Negri (UIN) Maulana Malik Ibrahim Malang
Untuk Memenuhi Salah Satu Persyaratan Dalam
Memperoleh Gelar Sarjana Komputer (S.Kom)

Oleh :
AINUN NAJIB
NIM. 15650053

JURUSAN TEKNIK INFORMATIKA
FAKULTAS SAINS DAN TEKNOLOGI
UNIVERSITAS ISLAM NEGERI MAULANA MALIK IBRAHIM
MALANG
2020

CENTRAL LIBRARY OF MAULANA MALIK IBRAHIM STATE ISLAMIC UNIVERSITY OF MALANG

LEMBAR PERSETUJUAN

ALGORITMA *SIMULATED ANNEALING* UNTUK PEMBANGKITAN *DATA TEST* SECARA OTOMATIS PADA PENGUJIAN PERANGKAT LUNAK

SKRIPSI

Oleh :

AINUN NAJIB
NIM.15650053

Telah diperiksa dan disetujui untuk Diuji

Tanggal: 27 November 2020

Pembimbing I	Pembimbing II
Fatchurrochman, M.Kom	Ajib Hanani, M.T
NIP. 1970031 200501 1 002	NIP. 19840731 20160801 1 076

Mengetahui,

Ketua Jurusan Teknik Informatika

Fakultas Sains dan Teknologi

Universitas Islam Negeri Maulana Malik Ibrahim Malang

Dr.Cahyo Crysdian

NIP.19740424 200901 1 008

CENTRAL LIBRARY OF MAULANA MALIK IBRAHIM STATE ISLAMIC UNIVERSITY OF MALANG

LEMBAR PENGESAHAN

ALGORITMA *SIMULATED ANNEALING* UNTUK PEMBANGKITAN *DATA TEST* SECARA OTOMATIS PADA PENGUJIAN PERANGKAT LUNAK

SKRIPSI

Oleh:
AINUN NAJIB
NIM. 15650053

Telah Dipertahankan di Depan Dewan Penguji
dan Dinyatakan Diterima Sebagai Salah Satu Persyaratan
untuk Memperoleh Gelar Sarjana Komputer (S.Kom)
Pada Tanggal 23 Desember 2020

Susunan Dewan Penguji			**Tanda tangan**
1. Penguji Utama	:	Fajar Rohman Hariri, M.Kom NIP. 19890515 201801 1 001	( )
2. Ketua Penguji	:	Agung Teguh Wibowo A, M.T NIP. 19890301 20180201 1 235	( )
3. Sekretaris Penguji	:	Fatchurrochman, M.Kom NIP. 1970031 200501 1 002	( )
4. Anggota Penguji	:	Ajib Hanani, M.T NIP. 19840731 20160801 1 076	( )

Mengetahui,
Ketua Jurusan Teknik Informatika
Fakultas Sains dan Teknologi
Universitas Islam Negeri Maulana Malik Ibrahim Malang

Dr. Cahyo Crysdian
NIP. 19740424 200901 1 008

CENTRAL LIBRARY OF MAULANA MALIK IBRAHIM STATE ISLAMIC UNIVERSITY OF MALANG

PERNYATAAN KEASLIAN TULISAN

Saya yang bertanda tangan dibawah ini :

Nama : Ainun Najib

NIM : 15650053

Fakultas/Jurusan : Sains dan Teknologi/Teknik Informatika

Judul Skripsi : Algoritma Simulated Annealing untuk Pembangkitan Data Test Secara Otomatis pada Pengujian Perangkat Lunak

Menyatakan dengan sebenarnya bahwa Skripsi yang saya tulis ini bener-benar merupakan hasil karya sendiri, bukan merupakan pengambilalihan data, tulisan atau pikiran orang lain yang saya akui sebagai hasil tulisan atau pikiran saya sendiri, kecuali dengan mencantumkan sumber cuplikan pada daftar pustaka.

Apabila dikemudian hari terbukti atau dapat dibuktikan Skripsi ini hasil jiplakan, maka saya bersedia menerima sanksi atas perbuatan tersebut.

Malang, 14 Desember 2020

Yang Membuat Pernyataan

Ainun Najib

Nim. 15650053

CENTRAL LIBRARY OF MAULANA MALIK IBRAHIM STATE ISLAMIC UNIVERSITY OF MALANG

HALAMAN MOTTO

"Dream Big, Work Hard, and Make it Happen"

CENTRAL LIBRARY OF MAULANA MALIK IBRAHIM STATE ISLAMIC UNIVERSITY OF MALANG

HALAMAN PERSEMBAHAN

الْحَمْدُ اللهِ رَبِّ الْعَالَمِينَ

Atas kehadirat Allah Subhanahu wa ta'ala, dengan mengucapkan Alhamdulillah penulis mempersembahkan sebuah karya untuk orang-orang yang sangat berarti

Terima kasih penulis ucapkan kepada kedua orang tua yang selalu tidak ada hentinya berusaha untuk memberikan yang terbaik untuk segala aspek dan tahap kehidupan penulis. Dengan jerih payah dan keringat beliau penulis bisa mencapai tahap ini dengan penuh rasa syukur telah dilahirkan dan dikaruniai ayah dan ibu sebagaimana Bapak Ruspandi dan Ibu Aminah. Tidak lupa Kakak yang selalu memberikan motivasi untuk segera lulus dan membuka tahap selanjutnya dalam hidup.

Terimakasih pula saya ucapkan untuk pembimbing yang telah membimbing dalam melakukan penilitian ini dan memberikan motivasi serta dorongan hingga penelitian ini dapat terselesaikan dengan lancar.

Tidak lupa terimaksih saya ucapkan kepada teman-teman satu perjuangan jurusan Teknik Informatika 2015 UIN Maulana Malik Ibrahim Malang yang telah menemani dan mengisi hari-hari selama 5 tahun terakhir. Dan kepada teman – teman yang selalu memberikan semangat dan bantuan dalam mengerjakan penelitian ini Alifiyah Pusaka Dewi Samudra, dan teman-teman kontrakan yang tiba - tiba terbentuk saya ucapkan terima kasih banyak.

Terima kasih untuk orang – orang yang tidak dapat disebutkan satu per satu yang telah memberikan motivasi, semangat dan doa sehingga penelitian skripsi ini dapat rampung dengan lancar.

CENTRAL LIBRARY OF MAULANA MALIK IBRAHIM STATE ISLAMIC UNIVERSITY OF MALANG

KATA PENGANTAR

بِسْمِ اللهِ الرَّحْمٰنِ الرَّحِيْمِ

Segala puji bagi Allah SWT, karena atas rahmat, hidayah dan karunia-Nya, penulis dapat menyelesaikan skripsi yang berjudul "Algoritma *Simulated Annealing* untuk Pembangkitan Data *Test* Secara Otomatis pada Pengujian Perangkat Lunak" sebagai salah satu syarat untuk memperoleh gelar sarjana pada Program Studi Teknik Informatika pada jenjang Strata-1 Universitas Islam Negeri Maulana Malik Ibrahim Malang.

Shalawat serta salam senantiasa selalu terlimpahkan kepada junjungan Nabi Muhammada SAW, keluarga dan para sahabat yang telah membimbing umat dari zaman kebodohan yaitu zaman jahiliyah menuju jalan yang diridzoi Allah SWT.

Penulis menyadari banyak keterbatasan yang penulis miliki, sehingga banyak pihak yang telah memberikan bantuan baik moril maupun materil dalam proses menyelesaikan penelitian ini. Maka dari itu dengan segenap kerendahan hati penulis mengucapkan terimakasih kepada :

1. Prof Dr H Abd. Haris, M.Ag selaku rektor UIN Maulana Malik Ibrahim Malang.
2. Dr. Sri Harini, M.Si selaku Dekan Fakultas Sains dan Teknologi UIN Maulana Malik Ibrahim Malang.

CENTRAL LIBRARY OF MAULANA MALIK IBRAHIM STATE ISLAMIC UNIVERSITY OF MALANG

3. Dr. Cahyo Crysdian selaku Ketua Jurusan Teknik Informatika Fakultas Sains dan Teknologi UIN Maulana Malik Ibrahim Malang.
4. Bapak H.Fatchurrochman, M.Kom selaku pembimbing I dan Bapak Ajib Hanani, M.T selaku pembimbing II yang senantiasa meluangkan waktu untuk membimbing, mengarahkan, dan memberi masukkan kepada penulis.
5. Seluruh Dosen Jurusan Teknik Informatika Fakultas Sain dan Teknologi UIN Maulana Malik Ibrahim Malang yang telah memberikan ilmu dan pengetahuan serta pengalaman yang sangat berharga dan bermanfaat.
6. Segenap civitas akademik Jurusan Teknik Informatika Fakultas Sains dan Teknologi UIN Maulana Malik Ibrahim Malang.
7. Kedua orang tua serta seluruh keluarga besar penulis yang selalu dengan senantiasa mendukung dan medoakan penulis.
8. Sahabar-sahabat seperjuangan Jurusan Teknik Informatika Fakultas Sains dan Teknologi UIN Maulana Malik Ibrahim Malang.

Penulis menyadari dalam karya ini masih banyak kekurangan. Oleh karena itu penulis selalu menerima segala kritik dan saran dari pembaca. Semoga karya ini dapat bermanfaat bagi seluruh pihak.

Malang, 23 Desember 2020

Penulis

CENTRAL LIBRARY OF MAULANA MALIK IBRAHIM STATE ISLAMIC UNIVERSITY OF MALANG

DAFTAR ISI

HALAMAN PENGAJUAN i
LEMBAR PERSETUJUAN ii
LEMBAR PENGESAHAN iii
PERNYATAAN KEASLIAN TULISAN iv
HALAMAN MOTTO v
HALAMAN PERSEMBAHAN vi
KATA PENGANTAR vii
DAFTAR ISI ix
DAFTAR GAMBAR xi
DAFTAR TABEL xii
ABSTRAK xiii
ABSTRACT xiv
الملخص xv
BAB I PENDAHULUAN 1
1.1 Latar Belakang 1
1.2 Identifikasi Masalah 5
1.3 Tujuan Penelitian 5
1.4 Manfaat Penelitian 5
1.5 Batasan Penelitian 5
1.6 Sistematika Penulisan 6
BAB II TINJAUAN PUSTAKA 8
2.1 Aplikasi Berorientasi Objek 8
2.2 *Software Testing* 8
2.3 *Automation Testing* 9
2.4 Pengujian Perangkat Lunak 10
2.5 Pengujian Jalur 12
2.6 Data Uji 13
2.7 Control Flow Graph (CFG) 13
2.8 Pengujian White Box 16
2.8.1 *Statement Coverage* 17

CENTRAL LIBRARY OF MAULANA MALIK IBRAHIM STATE ISLAMIC UNIVERSITY OF MALANG

2.8.2 *Branch Coverage* 19
2.8.3 *Conditiin Coverage* 19
2.8.4 *Path Coverage* 20
2.9 *Cyclomatic Complexity* 21
2.10 *Simulated Annealing* 24
2.11 Penelitian Terkait 28
BAB III METODOLOGI PENELITIAN 31
3.1 Analisis Data 31
3.2 Perancangan Sistem 31
3.2.1 Program Unit 33
3.2.2 *Parsing Program* 36
3.2.3 Membangun *Control Flow Graph* (CFG) 37
3.2.4 Pembentukan *Path* (Jalur) 40
3.2.5 *Simulated Annealing* 40
3.2.6 Pembangkitan *Data Test* 43
3.3 Pengujian *Simulated Annealing* Secara *Otomatis* 46
3.4 Spesifikasi Sistem 48
3.5 Evaluasi dan Validasi Hasil 48
BAB IV HASIL DAN PEMBAHASAN 49
4.1 Deskripsi Program 49
4.2 Hasil Pengujian 49
4.2.1 Data Uji Coba 50
4.2.2 Proses Pembuatan Aplikasi 51
4.2.3 Pengujian Aplikasi 53
4.3 Pembahasan 58
4.3.1 Hasil Uji Coba 58
4.3.2 Validasi Hasil 66
4.4 Integrasi Sains dan Islam 67
BAB V PENUTUP 70
4.1 Kesimpulan 70
4.2 Saran 70
DAFTAR PUSTAKA 71
LAMPIRAN 74

CENTRAL LIBRARY OF MAULANA MALIK IBRAHIM STATE ISLAMIC UNIVERSITY OF MALANG

DAFTAR GAMBAR

Gambar 2.1 Notasi Struktur Kontrol pada *Flow Graph* 14

Gambar 2.2 Contoh *flowchart* 15

Gambar 2.3 Transformasi *flowchart* ke *flowgraph* 16

Gambar 2.4 *Source Code* dan Grafik Kode Program 17

Gambar 3.1 Desain Sistem 32

Gambar 3.2 *Source Code benchmark Myers Triangle* 34

Gambar 3.3 *Source Code benchmark Michael Triangle* 35

Gambar 3.4 *Source Code benchmark Sthamer Triangle* 36

Gambar 3.5 *Source Code* Parsing Program 37

Gambar 3.6 CFG *Myers Triangle* 38

Gambar 3.7 *Source Code* Proses CFG 40

Gambar 3.8 *Flowchart* pembangkitan *data test*. 42

Gambar 3.9 *Source Code* Proses SA 47

Gambar 4.1 *Source Code* Implementasi SA. 53

Gambar 4.2 Tampilan Awal Program. 54

Gambar 4.3 Tampilan Halaman Utama. 54

Gambar 4.4 Tampilan Kolom Input. 55

Gambar 4.5 *Browsing* File *Java*. 56

Gambar 4.6 Tampilan Kolom Proses. 57

Gambar 4.7 *Hasil Output Proses Simulated Annealing* 58

Gambar 4.8 Hasil Analisa *Myers Triangle*. 59

Gambar 4.9 *Hasil Analisa Michael Triangle*. 61

Gambar 4.10 *Hasil Analisa Sthamer Triangle*. 64

Gambar 4.11 Hasil Validasi Data *Test*. 66

CENTRAL LIBRARY OF MAULANA MALIK IBRAHIM STATE ISLAMIC UNIVERSITY OF MALANG

DAFTAR TABEL

Tabel 2.1 Tabel Input Uji Kasus Pengujian *Path Coverage* 21

Tabel 3.1 Tabel Fungsi *Cost* atau *Fitness* 43

Tabel 3.2 Tabel Hasil Iterasi 46

Tabel 4.1 Tabel Hasil Data *Test Myers Triangle* 60

Tabel 4.2 Tabel Hasil Data *Test Michael Triangle* 63

Tabel 4.3 Tabel Hasil Data *Test Sthamer Triangle* 65

CENTRAL LIBRARY OF MAULANA MALIK IBRAHIM STATE ISLAMIC UNIVERSITY OF MALANG

ABSTRAK

Najib, Ainun. 2020. ***Algoritma Simulated Annealing Untuk Pembangkitan Data Test Secara Otomatis Pada Pengujian Perangkat Lunak.*** Skripsi. Jurusan Teknik Informatika Fakultas Sains Dan Teknologi Universitas Islam Negeri Maulana Malik Ibrahim Malang. Pembimbing : (I) Fatchurrochman, M.Kom. (II) Ajib Hanani, M.T.

Kata Kunci : *Simulated Annealing, Control Flow Graph, Branch Coverage*

Pengujian perangkat lunak memerlukan biaya yang mahal dan sering kali lebih dari 50% biaya keseluruhan dalam pengembangan perangkat lunak digunakan dalam tahapan ini. Untuk mengurangi biaya proses pengujian perangkat lunak secara otomatis dapat digunakan. Hal yang sangat penting dalam pengujian perangkat lunak secara otomatis adalah proses menghasilkan data tes. Pengujian secara otomatis yang paling efektif dalam menekan biaya adalah pengujian *branch coverage*. Salah satu metode yang banyak digunakan dan memiliki kinerja baik adalah algoritma *simulated annealing* (SA). Teknik pembangkitan data uji berbasis *algoritma simulated* annealing telah diaplikasikan secara luas agar waktu yang diperlukan dalam proses pengujian perangkat lunak dapat dikurangi. Data uji digunakan untuk mendeteksi adanya cacat perangkat lunak. Pada penelitian ini diusulkan algoritma *simulated annealing* sebagai pembangkit data uji untuk mengeksekusi semua cabang dalam sebuah program. *Control flow graph* dibangkitkan dari sebuah kode program untuk menggambarkan aliran kode program. Dengan data uji yang dapat diperoleh secara cepat maka cacat perangkat lunak dapat ditemukan lebih dini.

CENTRAL LIBRARY OF MAULANA MALIK IBRAHIM STATE ISLAMIC UNIVERSITY OF MALANG

ABSTRACT

Najib, Ainun. 2020. ***Algorithm Simulated Annealing For Automatic Generation Of Test Data in Software Testing***. Essay. Department of Informatics, Faculty of Science and Technology, Maulana Malik Ibrahim State Islamic University of Malang. Counselo: (I) H. Fatchurrohman, M.Kom. (II) Ajib Hanani, M.T.

Key Word : *Simulated Annealing, Control Flow Graph, Branch Coverage*

Software testing is expensive and often more than 50% of the total cost in software development is used at this stage. To reduce costs, automated software testing processes can be used. Of great importance in automated software testing is the process of generating test data. The most effective automatic test in reducing costs is the branch coverage test. One method that is widely used and has good performance is the simulated annealing algorithm (SA). The technique of generating test data based on the simulated annealing algorithm has been widely applied so that the time required in the software testing process can be reduced. Test data is used to detect software defects. In this research, the simulated annealing algorithm is proposed as a test data generator to execute all branches in a program. Control flow graph is generated from a program code to describe the program code flow. Control flow graph is generated from a program code to describe the program code flow. With test data that can be obtained quickly, software defects can be found early.

CENTRAL LIBRARY OF MAULANA MALIK IBRAHIM STATE ISLAMIC UNIVERSITY OF MALANG

الملخص

نجيب، عين. 2020. خوارزمية المقلدة التلدينة (simulated annealing (SA)) لإثارة بيانات الإختبار أووماتيكيا في إختبار البرمجيات. البحث العلمي. قسم المعلوماتية كلية العلوم والتكنولوجية جامعة مولانا مالك إبراهيم الإسلامية الحكومية مالانج. المشرف: 1) فتح الرحمن الماجستير، 2) أجب حنان الماجستير.

الكلمات الرئيسية: خوارزمية المقلدة التلدينة، القبضة التدفقة الرسمة البيانية، تغطية الفرع.

يحتاج إختبار البرمجيات المالية الغالية وطالما أكثر من 50% تكليفة جميعها في تنمية البرمجيات تستخدم هذه المرحلة. لإنقاص مالية عملية إختبار البرمجيات أوتوماتيكيا تستطيع ان تستخدم. الشأن الأهم في إختبار البرمجيات أوتوماتيكيا هو عملية حصيلة بيانات الإختبار. الإختبار الأوتوماتيكي الساري في إرهاق المالية هو إختبار تغطية الفرع (branch coverage). الإحدى من الطرق التي تستخدم كثيرا وتملك العمل الجيد هي خوارزمية المقلدة التلدينة (simulated annealing (SA)). تبنى الطريقة لإثارة بيانات الإختبار على خوارزمية المقلدة التلدينة طبقت واسعا لكي الوقت الذي يحتاج في عملية البرمجيات يستطيع ان ينقص. تستخدم بيانات الإختبار لإكتشاف كون شائبة البرمجيات. في هذا البحث، تقترح خوارزمية المقلدة التلدينة منشط بيانات الإختبار لإعدام جميع الفروع في البرنامج. تستفز القبضة التدفقة الرسمة البيانية (Control flow graph) من رمز البرنامج لتصوير جار الرمز البرنامج. ببيانة الإختبار التي تستطيع ان تنال سريعا فشائبة البرمجيات تستطيع ان تجد أأوّل.

CENTRAL LIBRARY OF MAULANA MALIK IBRAHIM STATE ISLAMIC UNIVERSITY OF MALANG

BAB I

PENDAHULUAN

Pada bab ini akan menjelaskan mengenai latar belakang penelitian, identifikasi masalah, tujuan penelitian, manfaat penelitian dan batasan penelitian. Latar belakang penelitian berisi penjelasan mengenai alasan peneliti mengangkat dan melakukan penelitian ini. Identifikasi masalah berisi sebuah pertanyaan yang didasarkan pada alasan dilakukan penelitian ini dan Tujuan Penelitian berisi tujuan dilakukannya penelitian ini. Batasan Penelitian berisi batasan pada penelitian agar penelitian tidak meluas dari fokus dilakukannya penelitian ini.

1.1 Latar Belakang

Pada zaman saat ini teknologi sudah sangat maju, terutama dibidang pembuatan perangkat lunak. Banyak perangkat lunak berkualitas yang sudah diciptakan. Sehingga permintaan dalam pembuatan perangkat lunak semakin banyak. Namun, kualitas perangkat lunak yang dikembangkan terkadang kurang baik karena tidak melalui proses pengujian perangkat lunak, padahal proses pengujian perangkat lunak merupakan tahap terpenting yang harus dilakukan.

Pengujian perangkat lunak merupakan tahap akhir dalam rangkaian pengembangan perangkat lunak. Pada tahap ini, dilakukan prosedur untuk memastikan tingkat kualitas dari perangkat lunak yang telah dibuat. Prosedur pengujian sistem tidak dapat dilakukan secara sembarangan, melainkan memiliki skenario atau kerangka uji yang dibuat oleh penguji sistem.

Proses pengujian perangkat lunak memerlukan biaya yang mahal (P McMinn, 2004), sering kali lebih dari 50% biaya total proses pengembangan perangkat lunak digunakan dalam proses ini. Untuk mengatasi hal tersebut

CENTRAL LIBRARY OF MAULANA MALIK IBRAHIM STATE ISLAMIC UNIVERSITY OF MALANG

pengujian perangkat lunak secara otomatis dapat dilakukan untuk mengurangi biaya (Díaz, Tuya, & Blanco, 2003), dan mendapatkan hasil pengujian yang lebih meyakinkan.

Salah satu upaya untuk menekan biaya proses pengujian dengan dilakukannya pengujian perangkat lunak secara otomatis, pembangkitan data tes merupakan hal yang paling utama. Pembangkitan data tes adalah proses identifikasi serangkaian uji kasus untuk memenuhi kecukupan data tes yang dipilih(Pachauri & Srivastava, 2013). Pembangkitan data tes menguraikan proses pencarian data tes terbaik untuk kriteria tes tertentu.

Pengujian perangkat lunak secara otomatis dapat dilakukan dengan menggunakan metode pencarian berbasis Metaheuristik atau yang dikenal dengan *Search Based Software Testing* (SBST) banyak digunakan dalam penelitian berdasarkan survey yang telah dilakukan oleh McMinn (McMinn, 2011). Metode ini dapat membangkitkan data tes dengan hasil cakupan yang tinggi dan mempunyai kemampuan dalam memperlihatkan kesalahan program yang diuji. Selain itu berdasarkan survey yang dilakukan Ali dkk (Ali dkk, 2010) menyatakan bahwa SBST lebih unggul dalam aspek efektifitas biaya dibandingkan dengan pembangkitan data tes secara tradisional. Beberapa algoritma SBST yang digunakan dalam membangkitkan data tes seperti Algoritma *Genetika* (GA), Algoritma *Ant Colony Optimization* (ACO), dan Algoritma *Simulated Annealing* (SA). Algoritma *Genetika* memiliki kinerja yang baik, satu permasalahan AG dalam menghasilkan data tes adalah ketiga target cabang dipilih memungkinkan tidak ada satupun individu yang memenuhi kriteria. Hal ini akan menyebabkan proses pencarian data tes memakan waktu lebih lama (Reza dkk., 2015).

CENTRAL LIBRARY OF MAULANA MALIK IBRAHIM STATE ISLAMIC UNIVERSITY OF MALANG

Srivastava dkk (P R Srivastava & Baby, 2010) mengadopsi algoritma ACO untuk menghasilkan urutan tes berdasarkan pengujian perangkat lunak, hal ini digunakan untuk memberikan hasil cakupan pengujian yang tinggi. Akan tetapi ACO memiliki kelemahan seperti pada algoritma SA, ACO memerlukan waktu yang lama dalam menghasilkan konvergensi. Algoritma genetika mempunyai kelebihan dalam menangani program dengan skala besar dan efektif dalam melakukan pencarian dibandingkan dengan metode pencarian yang lain (Yao & Gong, 2014).

Penggunaan algoritma *Simulated Annealing* untuk pengujian otomatis mempunyai kelebihan dalam memecahkan optimasi pada masalah program yang komplek. Begitu juga penelitian yang dilakukan oleh Patil dan Nikumb, bahwa *Simulated Annealing* (SA) adalah salah satu teknik paling fleksibel untuk menyelesaikan masalah kombinatorial yang sulit (Patil & Nikumb, 2012). Keuntungan utama dari *Simulated Annealing* adalah dapat diterapkan pada masalah besar terlepas dari kondisi diferensiabilitas, kontinuitas, dan konveksitas yang diperlukan dalam metode optimisasi konvensional.

Dalam kaitannya dengan Al-Qur'an, Allah SWT juga menjelaskan dalam firman-nya :

ٱلَّذِى خَلَقَ ٱلْمَوْتَ وَٱلْحَيَوٰةَ لِيَبْلُوَكُمْ أَيُّكُمْ أَحْسَنُ عَمَلًا ۚ وَهُوَ ٱلْعَزِيزُ ٱلْغَفُورُ ۝٢

Artinya : *"Yang menjadikan mati dan hidup, supaya Dia menguji kamu, siapa di antara kamu yang lebih baik amalnya. Dan Dia Maha Perkasa lagi Maha Pengampun". (QS Al-Mulk 67:02)*

CENTRAL LIBRARY OF MAULANA MALIK IBRAHIM STATE ISLAMIC UNIVERSITY OF MALANG

Tafsir Ibnu Katsir menjelaskan tentang *QS Al-Mulk* pada potongan ayat. (ٱلَّذِى خَلَقَ ٱلۡمَوۡتَ وَٱلۡحَيَوٰةَ لِيَبۡلُوَكُمۡ) "*Yang menjadikan mati dan hidup*" Ayat ini dijadikan oleh orang-orang yang berpendapat bahwa kematian adalah sesuatu yang wujud karena ia diciptakan (makhluk). Sedangkan makna ayat itu sendiri bahwa Allah telah mengadakan makhluk ini dari ketiadaan untuk menguji mereka, yakni untuk menguji siapakah diantara mereka yang paling baik amalnya. Sebagaimana yang difirmankan Allah Ta'ala,

كَيۡفَ تَكۡفُرُونَ بِٱللَّهِ وَكُنتُمۡ أَمۡوَٰتًا فَأَحۡيَـٰكُمۡۖ

"*Mengapa kamu kafir kepada Allah, padahal kamu tadinya mati, lalu Allah meghidupkan kamu,*" (*QS. Al-Baqarah 2:28*). Dengan demikian, keadaan pertama, yaitu ketiadaan sebagai maut (kematian). Sedangkan penciptaan disebut hayat (kehidupan). Oleh karena itu, Allah Ta'ala (لِيَبۡلُوَكُمۡ أَيُّكُمۡ أَحۡسَنُ عَمَلًا) "*Supaya Dia mengujimu, supaya diantara kamu yang lebik baik amalnya.*" Yakni, yang paling baik amalnya, sebagaimana yang dikatakan oleh Muhammad bin Ajlan. Dan Allah tidak Mengatakan "*Yang paling banyak amalnya.*"

Allah menguji hamba-Nya untuk mengetahui yang lebih baik amal diantara hamba-hamba-Nya. Sedangkan dalam penelitian ini, pengujian dimaksudkan sebagai sarana untuk mendapatkan *software* dengan kualitas yang baik.

Dalam penelitian ini, proses pengujian akan dilakukan menggunakan *benchmark* data program publik yang juga digunakan oleh peneliti lain seperti

CENTRAL LIBRARY OF MAULANA MALIK IBRAHIM STATE ISLAMIC UNIVERSITY OF MALANG

oleh Pachauri dan Srivastava (Pachauri & Srivastava, 2013). Penggunaan benchmark program publik memungkinkan orang lain dapat memvalidasi hasil penelitian yang dilakukan. Pada penelitian ini menggunakan tiga benchmark program yang banyak digunakan oleh peneliti, yaitu Myers Triangle, Michael Triangle dan Sthamer Triangle (Pachauri & Srivastava, 2013). Dengan pengujian yang dilakukan pada ketiga *benchmark* ini dapat diharapkan bisa mewakili pada sistem lainnya. Peneliti mengambil judul *Algoritma Simulated Annealing untuk Pembangkitan Data Test pada Pengujian Perangkat Lunak* sebagai penelitian yang akan dilakukan dalam skripsi ini.

1.2 Identifikasi Masalah

Berdasarkan latar belakang yang telah diuraikan, maka dalam penelitian ini akan membahas mengenai bagaimana ketepatan hasil pembangkitkan *data test* secara otomatis menggunakan algoritma *Simulated Annealing*?

1.3 Tujuan Penelitian

Berdasarkan identifikasi masalah, maka penelitian ini bertujuan untuk mengetahui ketepatan hasil pembangkitkan *data test* pada pengujian dengan menggunakan algoritma *Simulated Annealing* secara otomatis.

1.4 Manfaat Penelitian

Penelitian ini diharapkan agar dapat mempersingkat waktu penyusunan data test sehingga dapat menghemat waktu melakukan pengujian yang lainnya.

1.5 Batasan_Penelitian

Agar pembahasan penelitian ini tidak menyimpang, maka diperlukan batasan-batasan. Batasan-batasan dalam penelitian ini adalah :

CENTRAL LIBRARY OF MAULANA MALIK IBRAHIM STATE ISLAMIC UNIVERSITY OF MALANG

1. Menggunakan data set berupa *benchmark* data yang diambil dari http://shodhganga.inflibnet.ac.in/bitstream/10603/22181/18/18_appendix.pdf
2. Menggunakan bahasa pemrograman java.

1.6 Sistematika Penulisan

Laporan penelitian ini terdiri dari lima bab, dimana isi dari setiap bab terdiri dari :

BAB I : PENDAHULUAN

Berisi tentang latar belakang dari masalah yang akan diteliti, tujuan dan manfaat penelitian dari penelitian, batasan masalah pada penelitian, metodologi penelitian, serta sistematika penulisan laporan penelitian.

BAB II: TINJAUAN PUSTAKA

Bab ini berisi penjelasan mengenai penelitian yang telah dilakukan ataupun teori dasar dan data-data yang terkait dengan pembangkitan *data test* dalam pengujian perangkat lunak.

BAB III : METODE PENELITIAN

Bab ini berisi metode penelitian yang menjelaskan bagaimana penelitian ini dijalankan. Meliputi hasil analisa dan rincian langkah yang digunakan dalam pembangkitan *data test* dalam pengujian perangkat lunak dengan menerapkan metode *Simulated Annealing*.

BAB IV : HASIL DAN PEMBAHASAN

Bab ini berisi uji coba dari aplikasi yang telah dibuat dan dilakukan pembahasan secara terperinci terhadap proses pembangkit *data test*. Untuk

CENTRAL LIBRARY OF MAULANA MALIK IBRAHIM STATE ISLAMIC UNIVERSITY OF MALANG

selanjutnya membahas tentang hasil yang di dapat dari mengimplementasikan *Simulated Annealing*.

BAB V : KESIMPULAN DAN SARAN

Pada bab ini berisi kesimpulan dari penelitian yang telah dilakukan, saran dan kritik dari penelitian agar dapat dikembangkan pada penelitian selanjutnya.

CENTRAL LIBRARY OF MAULANA MALIK IBRAHIM STATE ISLAMIC UNIVERSITY OF MALANG

BAB II

TINJAUAN PUSTAKA

Pada bagian ini membahas tentang penelitian yang terkait, referensi-referensi penelitian dan konsep tentang teori yang digunakan dalam melakukan penelitian ini.

2.1 Aplikasi Berorientasi Objek

Aplikasi berorientasi objek adalah sebuah aplikasi yang dibangun dengan pemrograman berorientasi objek. Pemrograman berorientasi objek (*object-oriented programming*) merupakan paradigma pemrograman yang berorientasikan kepada objek (Rina, 2009). Semua data dan fungsi di dalam paradigma ini dibungkus dalam kelas-kelas atau objek-objek. Jika dibandingkan dengan logika pemrograman terstruktur, setiap objek dalam pemrograman berorientasi objek dapat menerima pesan, memproses data, dan mengirim pesan ke objek lainnya.

2.2 *Software Testing*

Software testing merupakan aktivitas yang dilakukan untuk mengevaluasi dan meningkatkan kualitas suatu produk, dengan cara mengidentifikasi *bug* serta permasalahan yang terdapat pada produk tersebut (Utting, 2007). *Software testing* berperan dalam menentukan ukuruan suatu project. Ukuran kualitas umumnya dibatasi oleh *correctness* (kebenaran), *completness* (kesempurnaan), *security* (keamanan). Namun ukuran kualitas menurut ISO *adalah reliability*, *efficiency*, *portability*, *maintainability*, *compatibility*, *usability*.

CENTRAL LIBRARY OF MAULANA MALIK IBRAHIM STATE ISLAMIC UNIVERSITY OF MALANG

2.3 *Automation Testing*

Automated testing merupakan proses pengujian yang digunakan untuk mempermudah proses dan dokumentasi pengujian, serta mengefektifkan proses pengeksekusian dan pengukuran pada pengujian (Novelia, 2008).

Otomatisasi testing akan sangat terasa manfaatnya (peningkatan efisiensi biaya dan efektifitas sumber daya) dalam *regression testing*. Terbatasnya waktu merupakan hambatan terbesar dalam melakukan *regression testing*, sehingga pada testing secara manual jumlah tes *case* untuk *regression testing* dibatasi hanya 10% dari jumlah tes case yang dilakukan pada awal testing. Berdasarkan pada studi yang dilakukan oleh Software Engineering Institute – Bellcore Study, terdapat kecenderungan terjadinya *defect*/*bug* baru setelah dilakukan perubahan atau perbaikan pada sistem (lebih dari 60%) atau error baru akan muncul setiap 6 baris kode dirubah.

Faktor- faktor yang mendukung berkembangnya pengujian perangkat lunak antar lain : penghematan biaya pengembangan, durasi pengujian yang dipersingkat, peningkatan kecermatan dalam pelaksanaan pengujian, dan peningkatan hasil pengujian (Rina, 2009).

Keuntungan pengujian otomatis :

a. Meningkatkan produktivitas

b. Menghemat uang

c. Meningkatkan kualitas perangkat lunak

d. Mengurangi waktu pengujian

e. Mendukung berbagai aplikasi

f. Meningkatkan cakupan pengujian

CENTRAL LIBRARY OF MAULANA MALIK IBRAHIM STATE ISLAMIC UNIVERSITY OF MALANG

g. Pengurangan pekerjaan yang berulang-ulang

2.4 Pengujian Perangkat Lunak

Pengujian perangkat lunak adalah proses atau serangkaian proses yang yang didesain untuk memastikan kode program berjalan dengan baik dengan melakukan apa yang sudah dirancang dan tidak melakukan apapun yang tidak diinginkan dalam rancangan tersebut (Reza, 2015). Sebuah perangkat lunak harus dapat diprediksi dan konsisten, hal tersebut dilakukan untuk menghasilkan perangkat lunak yang baik.

Menurut Reza (Reza, 2015) tujuan dari pengujian perangkat lunak adalah menemukan kesalahan di dalam program, dan pengujian yang baik adalah yang memiliki kemungkinan tinggi dalam menemukan kesalahan. Oleh karena itu dalam merancang dan menerapkan sistem/produk harus memenuhi kriteria testibility. Testibility memiliki arti seberapa mudah program komputer dapat diuji. Kriteria sebagai berikut menyebabkan perangkat lunak dapat diuji:

1. Operability

Jika sistem dirancang dan diimplementasikan dengan pemikiran untuk menghasilkan program yang berkualitas, maka akan relatif lebih sedikit dalam menemukan bugs yang akan menghalangi pelaksanaan pengujian. Hal ini akan menyebabkan proses pengujian perangkat lunak berjalan dengan baik.

2. Observability

Memberikan input pengujian yang berbeda-beda untuk menghasilkan output yang berbeda. Keadaan sistem dan variabel dapat terlihat dan dapat dipantau selama proses eksekusi berjalan. *Output* yang salah atau tidak sesuai

CENTRAL LIBRARY OF MAULANA MALIK IBRAHIM STATE ISLAMIC UNIVERSITY OF MALANG

dapat diidentifikasi dengan mudah. Kesalahan internal secara otomatis terdeteksi dan dilaporkan.

3. Controllability

Semua output yang mungkin dapat dihasilkan melalui beberapa kombinasi dari input, dengan format input/output konsisten dan terstruktur. Semua kode dieksekusi melalui beberapa kombinasi input. Keadaan dan variabel perangkat lunak dan keras dapat dikontrol secara langsung oleh penguji. Pengujian dapat dengan mudah diterapkan, dilakukan dengan otomatis, dan direproduksi.

4. Decomposability

Sistem perangkat lunak yang dibangun dari modul independen dapat diuji secara independen.

5. Simplicity

Program harus menunjukkan kesederhanaan fungsional (misalnya, set fitur adalah minimum yang diperlukan untuk memenuhi persyaratan); kesederhanaan struktural (misalnya, arsitektur modular untuk membatasi penyebaran kesalahan); dan kesederhanaan kode (misalnya, standar pengkodean diadopsi untuk kemudahan pemeriksaan dan pemeliharaan).

6. Stability

Perubahan perangkat lunak jarang terjadi, dikendalikan ketika terjadi perubahan, dan tidak membatalkan pengujian yang telah dilakukan. Perangkat lunak ini pulih dengan baik dari kerusakan.

7. Understandability

Desain arsitektur dan ketergantungan antara komponen internal, eksternal, dan shared dapat dipahami dengan baik. Dokumentasi teknis dapat langsung

CENTRAL LIBRARY OF MAULANA MALIK IBRAHIM STATE ISLAMIC UNIVERSITY OF MALANG

diakses, terorganisasi dengan baik, spesifik dan rinci, serta akurat. Perubahan desain harus dikomunikasikan kepada penguji.

2.5 Pengujian Jalur (Path)

Pengujian jalur atau path testing adalah strategi pengujian structural yang bertujuan untuk melatih setiap jalur eksekusi independen melalui komponen atau program. Jika setiap jalur independen dieksekusi, maka semua statement pada komponen harus dieksekusi paling tidak satu kali(Sommerville, 2003). Lebih jauh lagi, semua statement kondisional diuji untuk kasus true dan false. Pada proses pengembangan berorientasi objek, *test path* atau pengujian jalur dapat digunakan ketika menguji metode yang terkait dengan suatu program yang berorientasi objek ini.

Jumlah jalur yang dimiliki program biasanya sebanding dengan ukuran program. Namun jika model diintegrasikan kedalam sistem, pemakaian teknik pengujian structural menjadi tidak cocok. Teknik pengujian jalur dengan demikian paling cocok dipakai pada tahap pengujian unit dan pengujian modul pada proses pengujian terutama pengujian tahap awal sebelum program benar-benar jadi.

Dalam *test path* atau pengujian jalur, tidak menguji semua kombinasi jalur yang mungkin dilalui program. Untuk komponen yang tidak terdapat perulangan, maka tidak akan dieksekusi. Banyak kombinasi jalur yang akan dihasilkan pada program yang terdapat perulangan atau loop. Kekurangan atau kesalahan bisa terjadi ketika kombinasi jalur terbentuk bahkan ketika statement program telah dieksekusi paling tidak satu kali.

CENTRAL LIBRARY OF MAULANA MALIK IBRAHIM STATE ISLAMIC UNIVERSITY OF MALANG

Titik pokok dalam test path atau pengujian jalur merupakan graf alir atau flowgraph suatu program. Flow graph ini merupakan kerangka model yang mewakili semua jalur (path) yang ada dalam program.Flow graph terdiri dari node yang mewakili keputusan dan edge yang menunjukkan aliran control dengan diagram yang ekuivalen. Jika tidak ada statement goto pada program, penurunan flow graph termasuk pada proses yang sederhana. Statement sekuensial (assignment, pemanggilan prosedur, dan statement Input/Output) dapat diabaikan pada alur flow graph. Setiap percabangan yang mewakili statement kondisional (if-then-else atau case) ditunjukkan sebagai jalur yang terpisah. Sedangkan loop atau percabangan ditunjukkan dengan tanda panah yang kembali ke node kondisi loop (Sommerville, 2003).

2.6 Data Uji (*Test Data*)

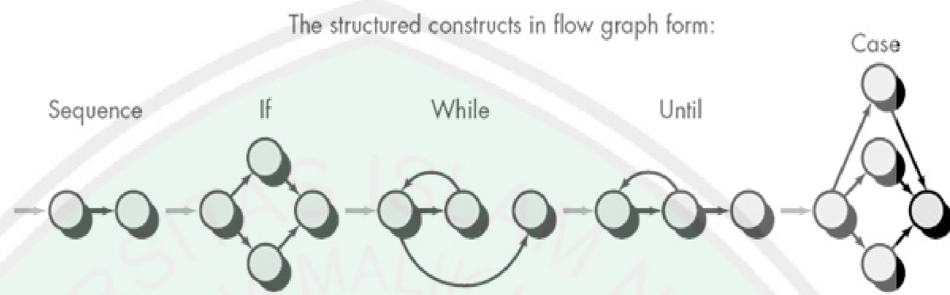
Test data atau data test adalah kumpulan informasi yang didapatkan dari source code, berisi inputan data agar fungsi dapat melakukan eksekusi dan menentukan arah path(Reza, 2015). Test data diperoleh dengan melihat setiap statement kondisi pada percabangan.

2.7 *Control Flow Graph* (CFG)

Flow Graph merupakan grafik yang digunakan untuk menggambarkan aliran kontrol dari sebuah program. Berbeda dengan *flowchart*, grafik pada *flow graph* tidak menggambarkan secara detail proses yang terjadi pada setiap blok notasi. Jenis notasi pada flowchart digambarkan secara berbeda (diamond, persegi panjang, jajar genjang, dst) untuk menggambarkan proses yang berbeda, sedangkan notasi pada flow graph hanya diwakili oleh sebuah notasi lingkaran. Dari penggunaannya, flowchart digunakan pada tahapan perancangan untuk

CENTRAL LIBRARY OF MAULANA MALIK IBRAHIM STATE ISLAMIC UNIVERSITY OF MALANG

menggambarkan logika dari program sedangkan flow graph digunakan pada tahapan pengujian yang berfokus pada penggambaran aliran kontrol sebuah program. Berikut ini adalah notasi struktur kontrol pada flow graph untuk menggambarkan sekuensial, seleksi, maupun perulangan:



Gambar 2.1 Notasi struktur kontrol pada flow graph

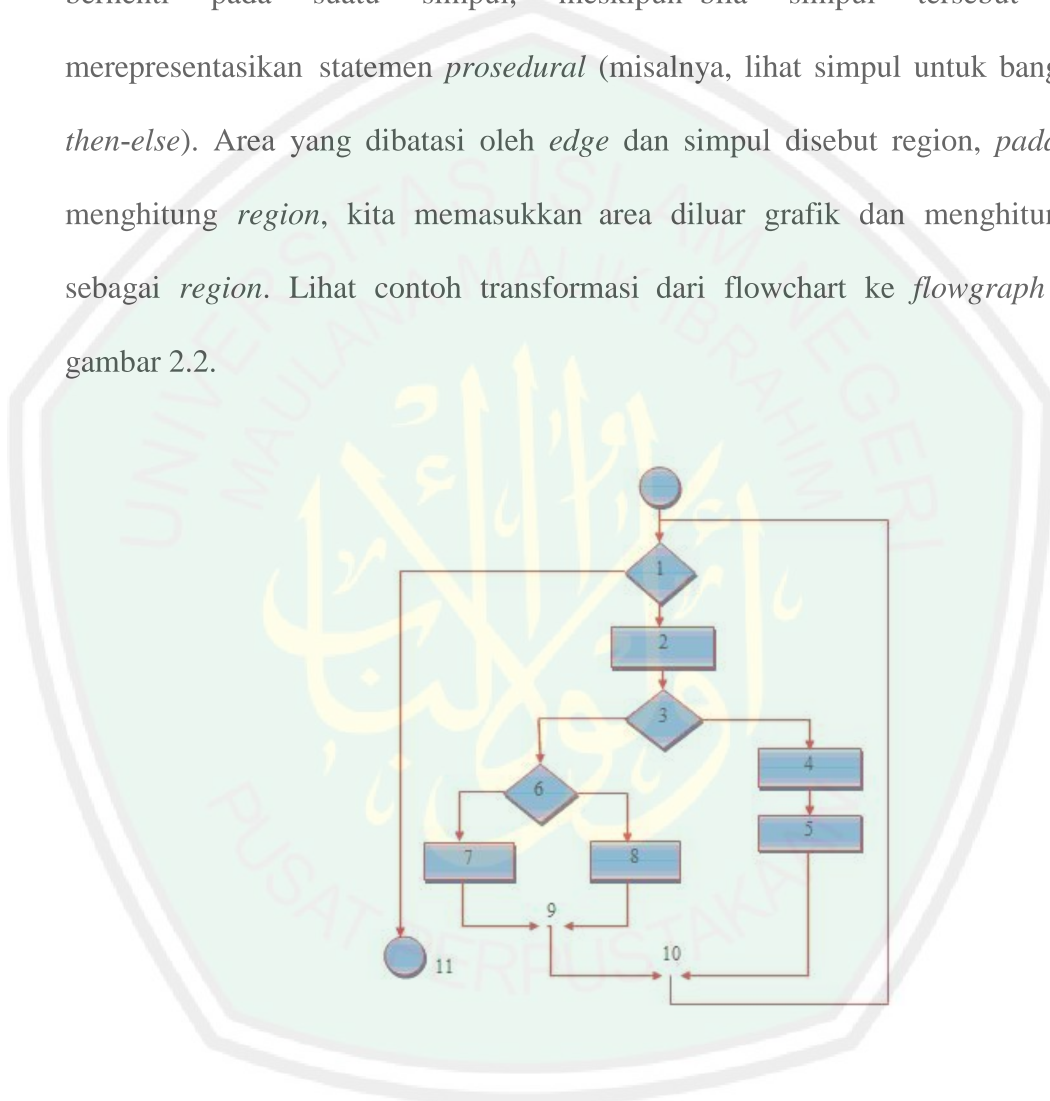
Notasi lingkaran disebut sebagai *flow graph node* yang digunakan untuk menggambarkan statement-statement berikut:

1. Satu atau lebih *statement* secara *sekuensial* yang dikelompokkan
2. Percabangan seleksi dari satu statement kedua pilihan statement (seleksi)
3. Penggabungan dua statement yang dilanjutkan pada satu statement yang sama (*merge*)

Sedangkan notasi garis panah disebut sebagai *edge* atau *link*, menggambarkan aliran kontrol. Setiap *edge* harus dihubungkan dari/kesebuah *node*, meskipun *node* tersebut tidak mewakili sebuah *statement* khusus. Area yang dibatasi oleh *node* dan *edge* disebut sebagai *region*. Secara sederhana, *flow graph* dapat dibuat dari grafik flowchart ataupun dari *pseudocode*/program *design language*/*source code* yang telah dibuat sebelumnya.

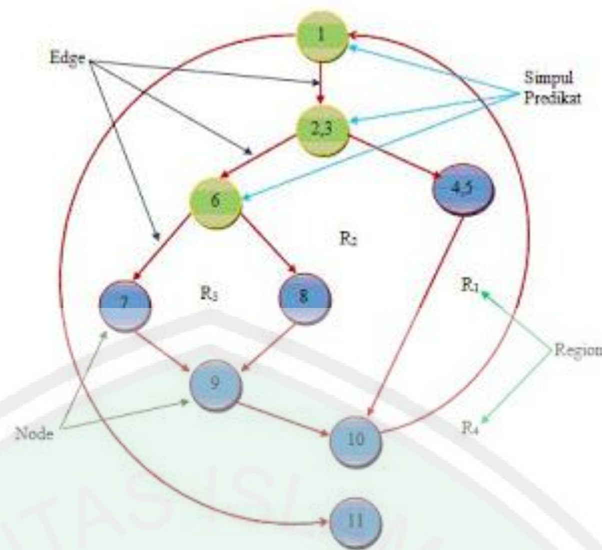
CENTRAL LIBRARY OF MAULANA MALIK IBRAHIM STATE ISLAMIC UNIVERSITY OF MALANG

Untuk menggambarkan *flowgraph* dapat menggunakan simpul *flowgraph* yang berbentuk lingkaran untuk merepresentasikan satu atau lebih *statemen prosedural*. Anak panah pada *flowgraph* disebut *edges* atau *links*, untuk merepresentasikan aliran kontrol dan analog dengan panah bagan alir. *Edge* harus berhenti pada suatu simpul, meskipun bila simpul tersebut tidak merepresentasikan statemen *prosedural* (misalnya, lihat simpul untuk bangun *if then-else*). Area yang dibatasi oleh *edge* dan simpul disebut region, *pada* saat menghitung *region*, kita memasukkan area diluar grafik dan menghitungnya sebagai *region*. Lihat contoh transformasi dari flowchart ke *flowgraph* pada gambar 2.2.



Gambar 2.2 Contoh flowchart (Sumber : Roger S. Pressman, 2002)

CENTRAL LIBRARY OF MAULANA MALIK IBRAHIM STATE ISLAMIC UNIVERSITY OF MALANG



Gambar 2.3 Transformasi flowchart ke flowgraph

(Sumber : Roger S. Pressman, 2002)

2.8 Pengujian White Box

Pengujian white box dikenal juga dengan pengujian struktural. Pengujian white box adalah pendekatan kasus pengujian untuk menguji struktur alur logika *source code* yang ada di dalam perangkat lunak. Tujuan dari pengujian ini adalah untuk menentukan apakah semua elemen logika dan data yang ada di dalam *source code* perangkat lunak berfungsi dengan baik. Terdapat beberapa teknik dalam pengujian white box, diantaranya *control flow testing*, *data flow testing* dan *mutation testing*. Diantara banyak pengujian white box *control flow testing* merupakan teknik yang paling populer dikarenakan sangat mudah dan efektif (Reza, 2015).

Untuk proses *control flow testing* diidentifikasi *path* yang ada di dalam *source code* program dan membuat uji kasus untuk mengeksekusi *path* tersebut. *Path* merupakan urutan pernyataan yang ada di dalam *source code* program yang terdapat masukan awal dan berakhir pada titik tertentu. Di dalam sebuah program memungkinkan terdapat banyak *path* dan mungkin semua path tersebut tidak dapat dieksekusi. Hal tersebut berhubungan jika terjadi penambahan *source code* program maka jumlah *path* akan semakin bertambah juga (Reza, 2015).

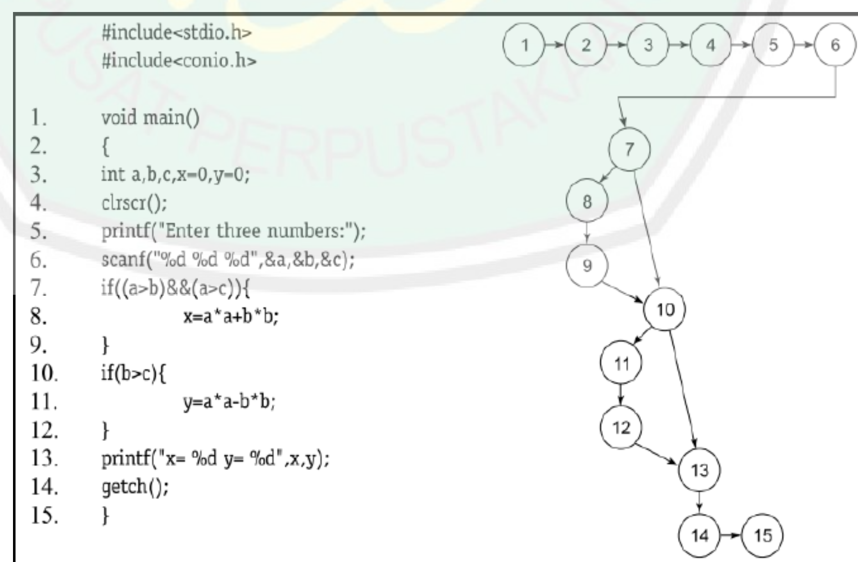
CENTRAL LIBRARY OF MAULANA MALIK IBRAHIM STATE ISLAMIC UNIVERSITY OF MALANG

Setiap *path* di dalam program memiliki porsi cakupan tersendiri, hal ini biasa disebut dengan istilah *coverage*. *Coverage* merupakan persentase source code di dalam program yang telah diuji dari total keseluruhan *source code* yang ada. Konsep menemukan kecukupan data tes untuk pemenuhan kriteria atau kebutuhan fitur tertentu dari *source code* harus dilaksanakan. Dalam prakteknya hal tersebut digunakan untuk menetapkan tujuan pengujian dan untuk mengembangkan serta mengevaluasi data tes yang diberikan, hal ini disebut sebagai *coverage analysis*. *Coverage* analysis biasa disebut juga dengan *coverage criteria* (Reza, 2015).

Teknik pengujian program menggunakan teknik *coverage* criteria memungkinkan memberikan kemudahan dalam mencapai efektifitas penyelesaian uji kasus. Ada beberapa teknik di dalam *coverage criteria*, diantaranya *statement coverage*, *branch coverage*, *condition coverage* dan *path coverage*. Pada bagian selanjutnya akan dijelaskan masing-masing dari *coverage criteria*.

2.8.1 *Statement Coverage*

Pada pengujian berdasarkan *statement coverage*, kita akan mengeksekusi semua pernyataan di dalam program untuk mencakup 100% cakupan pernyataan. Untuk mengetahui hal tersebut perhatikan Gambar 2.4 yang menunjukan contoh *source code* dan grafik kode program.



Gambar 2.4 Source Code dan Grafik Kode Program (Reza, 2015)

CENTRAL LIBRARY OF MAULANA MALIK IBRAHIM STATE ISLAMIC UNIVERSITY OF MALANG

Dengan memperhatikan Gambar 2.11 jika input nilai yang diberikan a=9, b=8, c=7, semua pernyataan di dalam kode program akan terpenuhi 100% pengujian *statement coverage* hanya dengan satu uji kasus. Total *path* yang terlewati adalah sebagai berikut:

1. 1-7, 10, 13-15
2. 1-7, 10-15
3. 1-10, 13-15
4. 1-15

Cyclomatic complexity dari grafik kode Gambar 2.4 adalah sebagai berikut:

$$V(G) = e - n + 2P = 16 - 15 + 2 = 3 \quad (2.1)$$

Dengan demikian, jumlah independent *path* ada 3 dengan path yang diberikan:

1. 1-7, 10, 13-15
2. 1-7, 10-15
3. 1-10, 13-15

Hanya dengan satu uji kasus yang diberikan memungkinkan untuk mencakup seluruh pernyataan yang ada di dalam program, tetapi tidak memungkinkan untuk mengeksekusi semua *path* yang ada, dan tidak juga mencakup semua *independent path* di dalam program tersebut.

Tujuan mencapai cakupan pernyataan 100% sulit dilakukan dalam kenyataannya. Sebagian dari program dapat melakukan keadaan yang tidak bisa diprediksi sebelumnya dan beberapa kondisi yang jarang mungkin terjadi. Hal ini mengakibatkan bagian program tertentu tidak dapat dijangkau karena kondisi tersebut sama sekali tidak dieksekusi.

CENTRAL LIBRARY OF MAULANA MALIK IBRAHIM STATE ISLAMIC UNIVERSITY OF MALANG

2.8.2 *Branch Coverage*

Branch coverage merupakan pengujian setiap cabang pernyataan yang menyatakan kondisi tertentu di dalam program. Setiap kondisi cabang dengan nilai "*true*" atau "*false*" akan dilakukan pengujian. Dengan memperhatikan Gambar 2.4 jika inputan diberikan dengan nilai a=9, b=8, c=7, cakupan pernyataan akan terpenuhi 100% dan *path* yang mengikutinya menunjukkan kondisi percabangan yang dilewati bernilai "*true*" semua dengan jalur:

Path = 1-15

Kita juga menginginkan kondisi "*false*" dapat dieksekusi, sehingga kita harus memasukkan uji kasus baru dengan nilai:

a=7, b=8, c=9 dengan jalur

Path = 1-7, 10, 13-15

Dari dua uji kasus yang diberikan, kedua cabang dengan dua kondisi cabang "*true*" dan dua kondisi cabang "*false*" dipastikan dapat terlewati. Hal ini menjamin pengujian *branch coverage* 100% terpenuhi. Sebagai catatan, bahwa terpenuhinya pengujian *branch coverage* tidak menjamin terpenuhinya 100% pengujian *path coverage*, akan tetapi akan menjamin terpenuhinya 100% cakupan untuk pengujian *statement coverage* (Reza, 2015).

2.8.3 *Condition Coverage*

Pengujian *condition coverage* lebih baik dibandingkan dengan pengujian *branch coverage* karena setiap kondisi di dalam program akan diuji minimal satu kali. Pengujian *branch coverage* dapat dicapai tanpa harus memperhatikan setiap kondisi yang ada (Reza, 2015).

Dengan memperhatikan Gambar 2.4 pada baris pernyataan ketujuh pada *source code* program. Pernyataan nomor tujuh tersebut mempunyai dua kondisi, yaitu kondisi (a>b) dan (a>c). Dari dua kondisi tersebut terdapat beberapa kemungkinan diantaranya:

1. Keduanya bernilai "*true*"

CENTRAL LIBRARY OF MAULANA MALIK IBRAHIM STATE ISLAMIC UNIVERSITY OF MALANG

2. Kondisi pertama “*true*”, kondisi kedua “*false*”

3. Kondisi pertama “*false*”, kondisi kedua “*true*”

4. Keduanya bernilai “*false*”

Jika a>b dan a>c, maka pernyataan baris ketujuh akan bernilai “*true*” (kemungkinan pertama). Kemudian jika a<b, secara otomatis kondisi kedua (a>c) tidak akan diuji dan pernyataan baris ketujuh bernilai “*false*” (kemungkinan ketiga dan keempat). Jika a>b dan a<c, pernyataan baris ketujuh untuk yang pertama bernilai “*true*” dan kedua bernilai “*false*” (kemungkinan kedua), hal ini akan menghasilkan nilai kondisi percabangan “*false*”. Nilai inputan yang didapatkan adalah:

1. a=9, b=8, c=7 (kemungkinan kondisi pertama, keduanya bernilai “*true*”)

2. a=9, b=8, c=10 (kemungkinan kondisi kedua, pertama bernilai “*true*”, dak kedua bernilai “*false*”)

3. a=7, b=8, c=9 (kemungkinan ketiga dan keempat dengan nilai pertama “*false*”, pernyataan baris ketujuh bernilai “*false*”)

Dapat diketahui, tiga uji kasus tersebut memenuhi dari empat kemungkinan yang ada untuk memastikan pelaksanaan setiap kondisi program terpenuhi.

2.8.4 *Path Coverage*

Dalam kriteria *coverage* ini, akan dilakukan pengujian setiap *path* yang ada di dalam program. Di dalam sebuah program memungkinkan terdapat banyak path termasuk kondisi pengulangan dan *return*. Hal tersebut mungkin menyebabkan tidak semua *path* dalam program dapat dieksekusi. Jika semua *path* di dalam program dapat tereksekusi semua, maka hal tersebut akan memberikan hasil bahwa program tersebut benar-benar baik. Jika semua *path* tidak dapat dicapai, setidaknya semua jalur independen harus dieksekusi.

Dengan memperhatikan Gambar 2.4, terdapat empat path untuk program tersebut, yaitu:

CENTRAL LIBRARY OF MAULANA MALIK IBRAHIM STATE ISLAMIC UNIVERSITY OF MALANG

1. 1-7, 10, 13-15

2. 1-7, 10-15

3. 1-10, 13-15

4. 1-15

Pelaksanaan semua *path* tersebut akan meningkatkan keyakinan tentang kebenaran program. Input untuk uji kasus yang diberikan dapat dilihat pada tabel 2.1:

Tabel 2.1 Input Uji Kasus Pengujian *Path Coverage*

No	ID Path	Paths	Data Test (Input)			Output yang diharapkan
			a	b	c	
1	Path 1	1-7, 10, 13-15	7	8	9	x=0, y=0
2	Path 2	1-7, 10-15	7	8	6	x=0, y=-15
3	Path 3	1-10, 13-15	9	7	8	x=130, y=0
4	Path 4	1-15	9	8	7	x=145, y=17

Pengujian *path coverage* menjamin jenis pengujian yang lain, yaitu *statement coverage*, *branch coverage* dan *condition coverage*. Namun demikian, terdapat banyak *path* yang ada di dalam program memungkinkan tidak dapat diuji. Jika semua *path* tidak dapat diuji, setidaknya semua *independent path* dapat dieksekusi yang merujuk pada dasar pengujian *path coverage* dengan cakupan yang wajar. *Independent path* dapat ditemukan dengan cara perhitungan *cyclomatic complexity*.

2.9 *Cyclomatic Complexity*

Cyclomatic complexity adalah metode pengukuran perangkat lunak yang memberikan pengukuran kuantitatif terhadap kompleksitas logika sebuah program. Pada konteks metode white box dengan teknik basic path, nilai yang dihitung dari cyclomatic complexity akan menentukan berapa jumlah jalur-jalur yang independen dalam basis set suatu program dan memberikan jumlah tes minimal yang harus dilakukan terhadap jalur independen untuk memastikan

CENTRAL LIBRARY OF MAULANA MALIK IBRAHIM STATE ISLAMIC UNIVERSITY OF MALANG

bahwa semua pernyataan yang sudah dibuat dalam jalur independen telah dieksekusi sekurangnya satu kali.

Jalur independen adalah jalur yang melalui program yang mengintroduksi sedikitnya satu rangkaian statemen proses baru atau suatu kondisi baru. Bila dinyatakan dengan teminologi flowgraph, jalur independen harus bergerak sepanjang paling tidak satu edge yang tidak dilewatkan sebelum jalur tersebut ditentukan.

Sebagai contoh, serangkaian jalur independen untuk grafik alir yang ditunjukkan pada gambar 2.3 adalah:

Path 1 = 1 – 11

Path 2 = 1 – 2 – 3 – 4 – 5 – 10 – 1 – 11

Path 3 = 1 – 2 – 3 – 6 – 8 – 9 – 10 – 1 – 11

Path 4 = 1 – 2 – 3 – 6 – 7 – 9 – 10 – 1 – 11

Perhatikan bahwa masing-masing jalur baru memperkenalkan sebuah edge baru.

Path 1 – 2 – 3 – 4 – 5 – 10 – 1 – 2 – 3 – 6 – 8 – 9 – 10 – 1 – 11 tidak dianggap jalur independen karena merupakan gabungan dari jalur-jalur yang sudah ditentukan dan tidak melewati beberapa edge baru.

Jalur 1, 2, 3, dan 4 yang ditentukan di atas terdiri dari sebuah basis set untuk flowgraph pada gambar 2.4. bila pengujian dapat dilakukan untuk memaksa adanya eksekusi dari jalur-jalur tesebut, maka setiap statemen pada

CENTRAL LIBRARY OF MAULANA MALIK IBRAHIM STATE ISLAMIC UNIVERSITY OF MALANG

program tersebut akan dieksekusi paling tidak satu kali dan setiap kondisi sudah akan dieksekusi pada sisi true dan false-nya. Perlu dicatat bahwa basis set tidaklah unik. Pada dasarnya, semua jumlah basis set yang berbeda dapat diperoleh untuk suatu desain prosedural yang diberikan.

Fondasi cyclomatic complexity adalah teori grafik, dan memberikan metriks perangkat lunak yang sangat berguna. Cyclomatic complexity dapat dihitung dalam salah satu dari tiga cara berikut:

1. Jumlah region grafik alir sesuai dengan cyclomatic complexity.
2. Cyclomatix complexity V(G) untuk grafik alir dihitung dengan rumus:

$$V(G) = E - N + 2 \quad (2.2)$$

Dimana:

E = jumlah edge pada grafik alir

N = jumlah node pada grafik alir

3. Cyclomatix complexity V(G) juga dapat dihitung dengan rumus:

$$V(G) = P + 1 \quad (2.3)$$

Dimana P = jumlah predicate node pada grafik alir

Dari Gambar di atas dapat dihitung cyclomatic complexity:

1. *Flowgraph* mempunyai 4 region
2. V(G) = 11 edge – 9 node + 2 = 4
3. V(G) = 3 predicate node + 1 = 4

CENTRAL LIBRARY OF MAULANA MALIK IBRAHIM STATE ISLAMIC UNIVERSITY OF MALANG

Jadi cyclomatic complexity untuk flowgraph pada gambar 2.3 adalah 4.

Yang lebih penting, nilai untuk V(G) memberi kita batas atas untuk jumlah jalur independen yang membentuk basis set. Jalur independen harus diuji untuk menjamin semua statemen dari program.

Nilai Cyclomatic Complexity yang tinggi menunjukkan prosedur kompleks yang sulit untuk dipahami, diuji dan dipelihara. Ada hubungan antara Cyclomatic Complexity dan resiko dalam suatu prosedur.

2.10 *Simulated Annealing*

Simulated Annealing (SA) berasal dari suatu makalah yang dipublikasikan oleh Metropolis tahun 1953. Jika kita memanaskan suatu materi keras hingga mencair dan kemudian mendinginkannya, maka sifat struktur materi tersebut bergantung pada tingkat pendinginan. Jika materi cair didinginkan secara perlahan, maka akan menghasilkan kristal yang berkualitas baik. Sebaliknya, jika materi cair didinginkan secara cepat, kristal-kristal yang terbentuk tidak akan sempurna. Algoritma yang diusulkan Metropolis mensimulasikan materi sebagai suatu sistem dari partikel-partikel. Algoritma tersebut mensimulasikan proses pendinginan yang secara bertahap menurunkan suhu sistem sampai konvergen pada keadaan beku dan stabil.

Pada tahun 1983, Kirk Patrick dan koleganya menggunakan ide dari algoritma *Metropolis* dan mengaplikasikannya pada permasalahan optimasi. Idenya adalah bagaimana menggunakan *simulated annealing* untuk mencari solusi-solusi yang layak dan konvergen pada solusi optimal. Kirk Patrick

CENTRAL LIBRARY OF MAULANA MALIK IBRAHIM STATE ISLAMIC UNIVERSITY OF MALANG

mengimplementasikan simulated annealing pada desain optimal hardware computer dan *Traveling Salesman Problem* (TSP).

Annealing adalah suatu pendinginan logam secara perlahan, setelah logam tersebut dipanaskan pada temperatur yang sangat tinggi. Proses pendinginan logam yang dipanaskan pada temperatur tinggi tersebut berlangsung secara perlahan-lahan,ketika penurunan temperatur berhenti, logam telah berada pada kondisi dengan energi yang sangat rendah.

Simulated annealing (SA) mensimulasikan proses *annealing* pada pembuatan materi yang terdiri dari butir kristal (*glassy*) atau logam. Tujuan dari proses ini adalah menghasilkan struktur kristal yang baik dengan menggunakan energi seminimal mungkin.

Ketika menyelesaikan masalah optimasi menggunakan *simulated annealing*, struktur dari sebuah zat akan mewakili penyusunan solusi dari sebuah masalah dan suhu digunakan untuk menentukan bagaimana dan kapan solusi baru dapat diperbarui dan diterima. Algoritma ini pada dasarnya adalah proses tiga langkah, yaitu memperbarui solusi, mengevaluasi kualitas solusi dan menentukan solusi yang diterima.

Pengimplementasian *simulated annealing* memerlukan sejumlah bilangan acak. Memilih pembangkit acak yang sesuai memerlukan pengetahuan khusus dari sebuah masalah. Pada dasarnya penting untuk menetapkan jumlah bilangan acak yang diperlukan dan menetapkan kecepatan dari pembangkit tersebut. Pembangkit acak tersebut nantinya akan menghasilkan solusi awal. Selain dengan menggunakan pembangkit acak, pembentukan solusi juga bisa dilakukan

CENTRAL LIBRARY OF MAULANA MALIK IBRAHIM STATE ISLAMIC UNIVERSITY OF MALANG

dengan algoritma lain seperti algoritma pencarian, dengan begitu kecepatan proses algoritma *simulated annealing* nantinya akan lebih baik (Suyanto, 2010).

Pada *simulated annealing* terdapat langkah untuk menentukan solusi yang baik, yaitu membentuk solusi baru kemudian mengevaluasi kualitas solusi. Pembentukan solusi baru dengan memodifikasi solusi terpilih atau terbaik dengan membangkitkan bilangan acak atau dengan menggunakan cara yang ditentukan sendiri. Pada pengaplikasiannya pembentukan solusi baru dapat dilakukan dengan pencarian tetangga dari solusi terpilih. Pencarian tetangga ini dapat berbeda-beda tergantung masalah yang dihadapi dan juga teknik pencariannya juga dapat ditentukan sendiri.

Pemilihan kondisi keadaan baru (*new state*) pada *Simulated Annealing* dilakukan secara acak dengan probabilitas tertentu. Jika *new statet* lebih baik dibandingkan *curren state,* maka SA selalu memilih *new state* tersebut. Tetapi, jika *new state* lebih buruk dari *curren state* (dalam hal ini *new state* memiliki biaya lebih besar dari daripada *curren state*), maka *new state* masih mungkin terpilih dengan probabilitas.

Fungsi probabilitas tersebut merepresentasikan distribusi Boltzmann dari energi dalam sistem termodinamika, sehingga didapat persamaan probabilitas dari level energi yang diberikan dalam sistem pada temperatur T. Pada fungsi tersebut, *k* adalah konstanta Boltzmann. Pada pratiktanya, konstanta Boltzmann tersebut seringkali tidak digunakan pada algoritma SA (Suyanto, 2010). Dengan demikian, probabilitas terpilihnya *new state* yang lebih buruk daripada *current state* adalah

$$\rho(\Delta E) = e^{-\Delta E/kT} > r \quad (2.4)$$

dimana

CENTRAL LIBRARY OF MAULANA MALIK IBRAHIM STATE ISLAMIC UNIVERSITY OF MALANG

ΔE = delta energi (menyatakan fungsi biaya atau evaluasi)

T = temperatur saat ini

r = bilangan acak antara 0 dan 1.

Pada rumus diatas terlihat bahwa jika temperatur T menurun maka probabilitas untuk menerima *new state* yang lebih buruk daripada *current state* juga menurun. Hal ini sama dengan pada yang terjadi pada *physical annealing,* dimana pergerakan temperatur menuju keadaan beku terjadi secara perlahan. Jika temperatur sama dengan nol, maka *new state* yang lebih buruk dari pada *current state* tidak akan terpilih. Dapat dikatakan bahwa untuk T = 0, algoritma SA adalah sama persis dengan *hill climbing*. Delta energi (perbedaan fungsi biaya antar *new state* dan *current state*) berbanding terbalik dengan besarnya probabilitasnya. Artinya, jika *new state* memiliki biaya yang jauh lebih besar dibandingkan *current state,* maka probabilitas terpilihnya *new state* akan semakin kecil.

Adapun algoritma dari *simulated annealing* adalah seperti dibawah ini:

1. inisialisasi

a. Solusi awal / solusi terpilih (S)

b.Suhu awal (T)

c. Bobot solusi awal C(S)

d. Reduksi (α)

2. Iterasi

a. Solusi baru (S`)

b. Bobot solusi baru C(S`)

c. Hitung delta energy $\Delta C = C(S`) - C(S)$

d. Jika $\Delta C \leq 0$

CENTRAL LIBRARY OF MAULANA MALIK IBRAHIM STATE ISLAMIC UNIVERSITY OF MALANG

i. (S) = (S`)

ii. C(S) = C(S`)

d. Jika $\Delta C > 0$

i. r = random(0,1)

ii. jiak $r < e^{\Delta(C)/T}$, (S) = (S`) dan C(S) = C(S`)

3. Temperatur (T)

2.11 Penelitian Terkait

Pada sub-bab ini akan dijelaskan beberapa penelitian yang digunakan sebagai referensi dalam melakukan penelitian ini. Penelitian-penelitian yang terkait dengan penelitian ini akan di-*review* sehingga dapat menunjang dan sebagai acuan dalam melakukan penelitian ini.

Hasil penelitian yang pernah dilakukan sehubungan dengan penelitian yang dilakukan dalam skripsi ini adalah :

Pachauri & Srivastava (Pachauri & Srivastava, 2013) mengusulkan model pencarian data tes dengan memanfaatkan metode pengurutan branch yang akan dicari, kemudian mengimplementasikan memory dan elitism. Metode perbaikan yang diusulkan adalah dalam proses pencarian data tes, yaitu dengan cara mengurutkan target yang akan dieksekusi. Selain mengurutkan target, peneliti mengusulkan teknik penyimpanan beberapa individu terbaik di dalam memory yang pada iterasi berikutnya akan menggantikan individu yang paling buruk. Penggunaan elitism juga diimplementasikan oleh peneliti untuk membandingkan dan menggantikan sebagian atau keseluruhan individu di dalam populasi yang dihasilkan untuk mendapatkan populasi terbaik. Hasil dari metode yang diusulkan menunjukkan peningkatan pencarian target yang lebih baik. Walaupun metode

CENTRAL LIBRARY OF MAULANA MALIK IBRAHIM STATE ISLAMIC UNIVERSITY OF MALANG

yang diusulkan memberikan hasil yang baik, masih terdapat kemungkinan untuk diperluas dan ditingkatkan performa dalam menghasilkan data tes.

Mandyartha, mengusulkan proses pembangkitan data tes menggunakan multi-populasi fuzzy adaptif pada algoritma genetika (Mandyartha, 2011). Teknik pembangkitan data uji berbasis *algoritma genetika* telah diaplikasikan secara luas agar waktu yang diperlukan dalam proses pengujian perangkat lunak dapat dikurangi. Data uji digunakan untuk mendeteksi adanya cacat perangkat lunak. Pada penelitian ini diusulkan algoritma genetika sebagai pembangkit data uji untuk mengeksekusi semua cabang dalam sebuah program. Control flow graph dibangkitkan dari sebuah kode program untuk menggambarkan aliran kode program, yang berisi cabang-cabang. Cabang target dipilih dari sub-sub populasi. Fuzzy adaptif digunakan untuk memperoleh parameter genetika secara dinamis berdasarkan kondisi pencarian. Pendekatan algoritma genetika yang diusulkan ini ketika diterapkan pada kumpulan program dengan jumlah cabang yang sangat banyak, telah ditunjukkan secara eksperimental bahwa lebih baik, dalam hal jumlah eksekusi dan waktu komputasi, dibandingkan dengan algoritma genetika multi-populasi yang parameter genetikanya bersifat statis. Dengan data uji yang dapat diperoleh secara cepat maka cacat perangkat lunak dapat ditemukan lebih dini.

Alshraideh, mengusulkan proses pembangkitan data tes menggunakan multiple-population pada algoritma genetika (Alshraideh, 2011). Dalam setiap tahap pencarian, tidak hanya satu kandidat target yang dicari, melainkan beberapa target sekaligus. Hal ini dilakukan untuk menghindari optimal lokal jika hanya satu target yang dicari. Multi-population juga akan mencegah terjadinya

CENTRAL LIBRARY OF MAULANA MALIK IBRAHIM STATE ISLAMIC UNIVERSITY OF MALANG

premature *convergence* yang akan mengakibatkan hasil pencarian tidak bisa beragam. Metode yang diimplementasikan menunjukan pencarian terhadap target menjadi lebih singkat, jumlah eksekusi yang diperlukan untuk cakupan target lebih banyak, dan lebih efektif dalam proses pencarian jika dibandingkan dengan yang hanya menggunakan single-population.

Yao & Gong (Yao & Gong, 2014) mengusulkan pembangkitan data tes menggunakan algoritma *genetika* dengan memanfaatkan individual sharing di dalam multiple-population. Metode yang diusulkan Yao & Gong berbeda dengan multiple-population tradisional. Tujuan utama dari strategi yang diusulkan adalah untuk memperluas jangkauan pencarian setiap populasi dengan berbagi individu, sehingga dapat meningkatkan efisiensi algoritma.

Fitriani & Hermadi (Fitriani & Hermadi, 2018) membangun sebuah aplikasi untuk membangkitkan kemungkinan jalur-jalur dari sebuah program yang dapat dijadikan dasar untuk membangkitkan data uji agar data uji yang digunakan untuk pengujian dapat mewakili semua kemungkinan. Selain itu, aplikasi ini juga dapat melakukan penyisipan tag-tag sebagai instrumentasi ke dalam kode program secara otomatis untuk memonitor jalur mana yang dilalui ketika diberikan masukan data uji.

CENTRAL LIBRARY OF MAULANA MALIK IBRAHIM STATE ISLAMIC UNIVERSITY OF MALANG

BAB III

METODOLOGI PENELITIAN

Pada bab ini akan dibahas mengenai beberapa hal, yaitu tahapan penelitian yang akan dilakukan, kebutuhan sistem yang akan dibuat, dan penyelesaian masalah pembangkitan *data test* pada pengujian perangkat lunak menggunakan metode *simulated annealing*.

3.1 Analisis Data

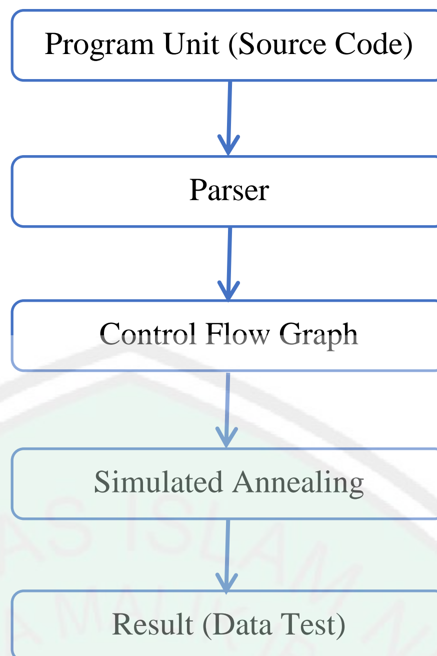
Pada bagian ini akan menjelaskan langkah-langkah yang akan ditempuh dalam menyelesaikan skripsi. Tahap design dan inplementasi aplikasi tidak menjamin suatu aplikasi terbebas dari kesalahan. Ditemukan banyaknya pengembangan yang dilakukan pada suatu aplikasi berbasis objek, sehingga aplikasi berbasis objek perlu dilakukan pengujian.

Pembangkitan data uji (*data test*) dilakukan dengan beberapa cara, sejauh ini pembangkitan data uji pada sebuah aplikasi pada satu tahap coding atau tahap pengujian. Pembangkitan data uji pada aplikasi berbasis objek dapat dilakukan pada saat tahap coding dengan memanfaatkan model search based testing.

3.2 Perancangan Sistem Aplikasi

Bagian ini akan menjelaskan tahapan yang harus dilalui dalam sistem yang akan dibangun. Langkah desain sistem ini dapat dilihat pada Gambar 3.1 di bawah ini :

CENTRAL LIBRARY OF MAULANA MALIK IBRAHIM STATE ISLAMIC UNIVERSITY OF MALANG



Gambar 3.1 Desain Sistem

Pembuatan aplikasi pembangkitan data tes otomatis ini akan menggunakan *control flow graph* dan *simulated annealing*. Pada saat implementasi masing-masing proses mempunyai tugas tersendiri pada saat pembangkitan data tes. Tetapi, kedua proses tersebut saling mendukung untuk pembangkitan data tes.

Control flow graph akan difokuskan untuk pembentukan jalur atau path sebagai inputan ke *simulated annealing*. Sedangkan algoritma *simulated annealing* sendiri difokuskan untuk proses menghasilkan data tes berdasarkan inputan dari hasil *control flow graph*.

Pada tahap perancangan sistem untuk membangkitkan *data test*, sebelum proses membangun *control flow graph* dan metode *simulated anneling*, mengacu pada gambar 3.1 bisa dijelaskan langkah pertama adalah membuat instrumen program yang kan diuji (program unit) sebagai masukan (*input*). Instrumen ini

CENTRAL LIBRARY OF MAULANA MALIK IBRAHIM STATE ISLAMIC UNIVERSITY OF MALANG

yang nantinya akan berpengaruh pada penentuan nilai fitness. Masukan (*input*) berupa fungsi dari program unit (*source code*) kemudian dianalis menggunakan *control flow graph* (CFG) yang menggambarkan aliran (*path*) kode program. Setelah aliran (*path*) kode program diketahui kemudian dibangkitkan data uji dengan metode algoritma *simulated annealing*. Jalur-jalur yang tergambar pada CFG dapat digunakan untuk menentukan jalur yang harus dipenuhi agar sebuah cabang target dapat dipenuhi (dieksekusi).

3.2.1 Program Unit

Di dalam penelitian ini menggunakan *benchmark* data program publik yang juga digunakan oleh peneliti lain seperti oleh pachauri dan srivastava. Penggunaan *benchmark* program publik memungkinkan orang lain dapat memvalidasi hasil penelitian yang dilakukan. Pada penelitian ini menggunakan tiga *benchmark* program yang banyak digunakan oleh peneliti, yaitu *Myers Triangle*, *Michael Triangle* dan *Sthamer Triangle*. *Benchmark* program tersebut didapatkan dari situs **http://shodhganga.inflibnet.ac.in/bitstream/10603/22181/18/18_appendix.pdf**.

Source code lengkap benchmark program yang digunakan dapat dilihat pada lampiran. Berikut ini merupakan informasi *benchmark* program yang digunakan:

1. *Myers Triangle*

Program ini menggolongkan segitiga atas dasar sisi input sebagai non-segitiga atau segitiga, yaitu, sama kaki, sama sisi atau sisi tak sama panjang. Diberikan tiga input nilai di dalam parameter yang semuanya mewakili sisi segitiga. *Control*

CENTRAL LIBRARY OF MAULANA MALIK IBRAHIM STATE ISLAMIC UNIVERSITY OF MALANG

flow graph yang dihasilkan memiliki 14 node dengan predikat *node* berjumlah 6. Memiliki kondisi kesetaraan dengan operator DAN, yang membuat cabang sulit untuk dilewati. Berikut *source code* dari *benchmark* data *Myers Triangle*.

```
public static void Triangle(double A, double B,
double C){
String type="";
double perimeter;
if (A > 0 && B > 0 && C > 0){
perimeter = A + B + C;
if (((2 * A) < perimeter) && ((2 * B) < perimeter) &&
((2 * C) < perimeter)){
if (A == B) {
if (B == C) {
type = "equilateral";
}else{
type = "isosceles";}
}else{
if (A == C) {
type = "isosceles";
}else{
if (B == C){
type = "isosceles";
}else{
type = "scalene";}
}}
}else{
type = "notriangle";}
}
else
{
type = "notriangle";}
}
```

Gambar 3.2 *source code benchmark Myers Triangle*

2. *Michael Triangle*

Program ini juga menggolongkan segitiga atas dasar sisi input sebagai non-segitiga atau segitiga, yaitu, sama kaki, sama sisi atau sisi tak sama panjang. Dibutuhkan tiga input nilai dengan semua dari ketiganya mewakili sisi segitiga tetapi dengan kondisi predikat yang berbeda. *Control flow graph* yang dihasilkan memiliki 26 *node* dengan jumlah predikat *node* 11. Berikut *source code* dari *benchmark* data *Michael Triangle*.

CENTRAL LIBRARY OF MAULANA MALIK IBRAHIM STATE ISLAMIC UNIVERSITY OF MALANG

```
int triangle(double i, double j, double k){
        int tri = 0;
        if ((i<=0) || (j<=0) || (k<=0))
            return 4;
        else if (i==j)
            tri += 1;
        else
        if (i==k)
            tri += 2;
        else
        if (j==k)
            tri += 3;
        else
        if (tri==0){
            if ((i+j<=k) || (j+k<=i) || (i+k<=j))
                tri = 4;
            else
                tri = 1;
            return tri;
        }else{
            if (tri>3)
                tri=3;
            else if ((tri==1) && (i+j>k))
                tri = 2;
            else if ((tri==2) && (i+k>j))
                tri = 2;
            else if ((tri==3) && (j+k>i))
                tri = 2;
            else
                tri = 4;}
        return tri;}
```

Gambar 3.3 *source code benchmark Michael Triangle*

3. *Sthamer Triangle*

Seperti *benchmark* program sebelumnya, program ini juga menggolongkan segitiga atas dasar sisi input sebagai non-segitiga atau segitiga, yaitu, sama kaki, sama sisi, segitiga sudut kanan atau sisi tak sama panjang. *Control flow graph* yang dihasilkan memiliki 29 node dengan jumlah predikat node 13. Program ini memiliki kondisi persamaan dengan operator DAN dan operator relasional yang kompleks. Berikut *source code* dari *benchmark* data *Sthamer Triangle*.

CENTRAL LIBRARY OF MAULANA MALIK IBRAHIM STATE ISLAMIC UNIVERSITY OF MALANG

```
static int triangle3(double a, double b, double c) {
        double perim;
        int tri_kind;
        if (a > 0) {
            if (b > 0) {
                if (c > 0) {
                perim = a + b + c;
                if ((2 * a) < perim) {
                if ((2 * b) < perim) {
                if ((2 * c) < perim) {
                if (a == b) {
                if (b == c) {
                   tri_kind = 2;
                   } else {
                   tri_kind = 1;}
                   } else {
                if (a == c) {
                   tri_kind = 1;
                   } else {
                if (b == c) {
                   tri_kind = 1;
                if ((a * a + b * b) == (c * c)) {
                   tri_kind = 3;
                   } else {
                if ((b * b + c * c) == (a * a)) {
                   tri_kind = 3;
                   } else {
                if ((a * a + c * c) == (b * b)) {
                   tri_kind = 3;
                   } else {
                   tri_kind = 4;}
                   }}}}
                   } else {
                   tri_kind = 0;}
                   } else {
                   tri_kind = 0;}
                   } else {
                        tri_kind = 0;}
                } else {
                    tri_kind = 0;}
            } else {
                tri_kind = 0;}
        } else {
            tri_kind = 0;}
        return tri_kind;
    }
```

Gambar 3.4 *source code benchmark Sthamer Triangle*

3.2.2 *Parsing Program*

Parsing adalah tahapan dimana sintaks-sintaks dari *query* akan di cek untuk menentukan apakah *query* tersebut sudah dirumuskan sesuai dengan aturan-

CENTRAL LIBRARY OF MAULANA MALIK IBRAHIM STATE ISLAMIC UNIVERSITY OF MALANG

aturan *sintaks* (aturan-aturan grammar) dari bahasa *query*. Setelah mengalami proses *parsing* di dalam *parser*, maka kemudian *query* tersebut diproses di dalam optimizer untuk mendapatkan rencana eksekusi. Proses parsing merupakan tahapan analisis sintaksis yang berguna untuk memeriksa urutan kemunculan token. Parsing dapat diprogram dengan tangan atau mungkin (semi-) pen-generate otomatis(pada beberapa bahasa pemrograman) dengan bantuan tool. Di dalam mengimplementasikan *parsing* ke dalam program perlu diperhatikan tiga hal, yaitu:

1. Rentang waktu eksekusi.
2. Penanganan kesalahan.
3. Penanganan kode.

Berikut *source code* untuk *parsing* data uji.

```
public Parser(String path) throws IOException {
      ArrayList<String> lines = new ArrayList<>();
      FileReader fileReader =
        new FileReader(path);
            BufferedReader bufferedReader =
                new BufferedReader(fileReader);
            String line;
                  int i=0;
            while((line = bufferedReader.readLine())
!= null) {
                lines.add(line);
                text_output.append(i+":"+line+"\n");
                i++;
                  }
            bufferedReader.close();
                  Graph2 graph = new Graph2(lines);
                  System.out.println();
                  graph.buildGraph();
}
```

Gambar 3.5 *source code parsing* Program

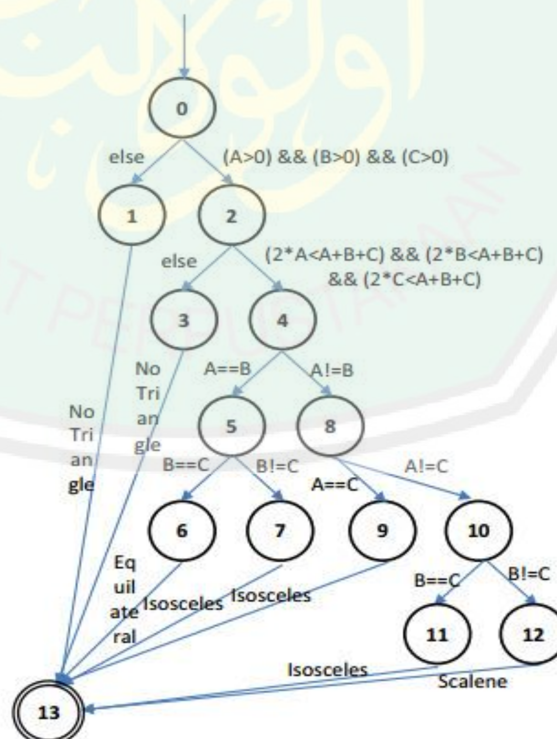
3.2.3 Membangun *Control Flow Graph* (CFG)

Control Flow Graph (CFG) merupakan graph berarah yang merepresentasikan aliran dari sebuah program. Setiap CFG terdiri dari *nodes* dan

CENTRAL LIBRARY OF MAULANA MALIK IBRAHIM STATE ISLAMIC UNIVERSITY OF MALANG

edges. *Nodes* merepresentasikan perintah. Sedangkan *edges* merepresentasikan kontrol transfer antar *nodes*. *Node* dapat menggambarkan beberapa statement tergantung dengan kondisi, baik itu secara sekuensial, perulangan dan lain-lain, dapat berupa percabangan.

Fungsi utama dari CFG adalah mengambarkan aliran kode program mulai awal program hingga akhir program. CFG dari fungsi program pada gambar 3.2 digambarkan oleh gambar 3.3. Berdasarkan jalur yang tergambar pada CFG tersebut, untuk mengeksekusi cabang target yaitu “if(2*A<A+B+C && 2*B<A+B+C && 2*C<A+B+C)” maka cabang “if(A>0 && B>0 && C>0)” harus dipenuhi, artinya cabang “if(2*A<A+B+C && 2*B<A+B+C && 2*C<A+B+C)” control dependence terhadap cabang “if(A>0 && B>0 && C>0)”. “2*A<A+B+C && 2*B<A+B+C && 2*C<A+B+C” dan “A>0 && B>0 && C>0” disebut juga predikat cabang.



Gambar 3.6 CFG Myers Triangle (Sumber : Manikumar dkk, 2016)

CENTRAL LIBRARY OF MAULANA MALIK IBRAHIM STATE ISLAMIC UNIVERSITY OF MALANG

Berdasarkan *graph* yang telah terbentuk dari kode program *myers triangle*, dapat dilihat pada gambar 3.4 bahwa *node* yang terbentuk adalah 14 dan jumlah *edge* yang terbentuk adalah 19. Sehingga hasil perhitungan *cyclomatic complexity* dapat dilihat pada persamaan dibawah ini.

$$V(G) = (E - N) + 2 \quad (3.1)$$

$$V(G) = (19 - 14) + 2 = 7$$

Hasil perhitungan *cyclomatic complexity* dari kode program *myers triangle* adalah 7 yang menunjukkan jumlah jalur dasar yang akan terbentuk.

```
public Nodes buildGraph(){
      Nodes root = new Nodes(nodeName,0);
      nodeName++;
      buildChild(root, true);
      ArrayList<Nodes>visited = new ArrayList<>();
      HashMap<Character, ArrayList<Character>> table
= new HashMap<>();
      visitTree(root,visited,table);
      int size = table.keySet().size();
      int arr[][] = new int[size][size];
      create2Darray(table,arr);
      int cyclomaticComplexity = 0;
      cyclomaticComplexity =
findPaths(root,"",cyclomaticComplexity);
      text_cfg.append("\n");
      text_cfg.append("The cyclomatic complexity:
"+cyclomaticComplexity);
      return root;
      }
private int findPaths(Nodes root, String string, int
cyclomaticComplexity) {
      boolean isLoop = false;
      if(string.contains(String.valueOf(
root.nodeName))) isLoop = true;
      string
=string.concat(String.valueOf(root.nodeName));
      if(root.childs.isEmpty()){
      text_cfg.append(string+"\n");
      cyclomaticComplexity++;
      }
      else string = string.concat("-->");
      ArrayList<Nodes>childs = root.childs;
      Nodes current;
```

CENTRAL LIBRARY OF MAULANA MALIK IBRAHIM STATE ISLAMIC UNIVERSITY OF MALANG

```
      for(int i=0;i<childs.size();i++){
            current = childs.get(i);
            if(!string.contains(String.valueOf(
current.nodeName)) || !isLoop)
            cyclomaticComplexity =
Math.max(cyclomaticComplexity,
findPaths(current,string,cyclomaticComplexity));
                }
            return cyclomaticComplexity;
}
```

Gambar 3.7 *source code* proses CFG

Source code diatas merupakan proses untuk membentuk sebuah jalur atau *path* yang digunakan untuk mengetahui jalur *node* mana yang benar dan jalur *node* mana yang salah.

3.2.4 Pembentukan *Path* (Jalur)

Masukan (input) berupa fungsi dari program kemudian dianalisis menggunakan control flow graph (CFG) yang menggambarkan aliran (path) kode program. Berikut merupakan semua kemungkinan jalur yang akan yang dapat dijadikan sebagai dasar dalam pembangkitan data uji, kemungkinan jalur direpresentasikan dalam bentuk urutan nomor node :

Path 1 = 0 – 1 – 13

Path 2 = 0 – 2 – 3 – 13

Path 3 = 0 – 2 – 4 – 5 – 6 – 13

Path 4 = 0 – 2 – 4 – 5 – 7 – 13

Path 5 = 0 – 2 – 4 – 8 – 9 – 13

Path 6 = 0 – 2 – 4 – 8 – 10 – 11 – 13

Path 7 = 0 – 2 – 4 – 8 – 10 – 12 – 13

3.2.5 *Simulated Annealiang*

Algoritma *simulated annealing* merupakan algoritma optimasi yang membutuhkan solusi awal yang nantinya dari solusi awal tersebut akan dibentuk solusi baru. Pembentukan solusi baru bisa dilakukan dengan menggunakan

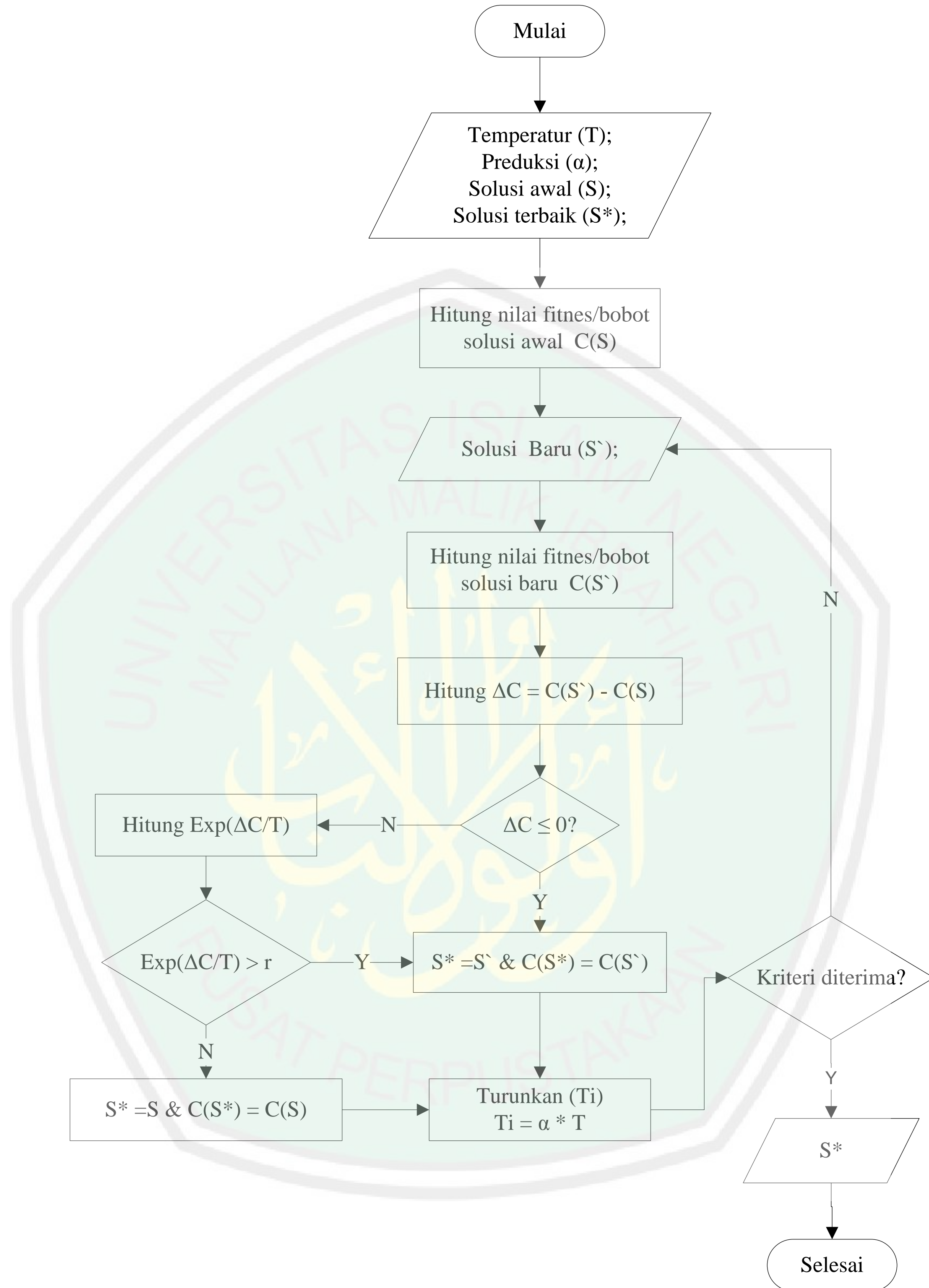
CENTRAL LIBRARY OF MAULANA MALIK IBRAHIM STATE ISLAMIC UNIVERSITY OF MALANG

bilangan acak atau dengan teknik-teknik sendiri yang sudah ditentukan. Setelah mendapatkan solusi awal, dari solusi awal tersebut akan dihitung bobot dari solusi awal, kemudian dibandingkandengan bobot dari solusi baru. Jika bobot dari solusi baru lebih baik maka solusi baru dijadikan solusi terbaik (*best so far*). Jika solusi baru mempunyai bobot yang lebih buruk kemungkinan terpilihnya solusi baru juga masih ada dengan menggunakan perhitungan probabilitas. Setelah solusi terbaik terpilih, temperatur atau suhu diturunkan dengan pereduksi suhu yang sudah ditentukan sebelumnya.

Langkah tersebut akan diulang terus menerus sampai tujuan tercapai. Perulangan proses di atas di ulang hingga nilai fitness atau bobot dari sebuah solusi bernilai 0. Tidak hanya itu, beberapa referensi mengatakan perulangan proses tersebut dapat dihentikan jika nilai fitness atau bobot dari solusi baru nilainya sama berturut-turut. Hal ini berarati solusi baru yang dibentuk tidak ada perubahan karena nilai fitness atau bobot tidak berubah.

Pada aplikasi kali ini solusi awal tidak dibangkitkan secara acak melainkan dengan menggunakan solusi random yang dibentuk untuk mendapatkan nilai fitness atau bobot pada langkah diatas. Solusi awal tersebut yang akan digunakan untuk membentuk solusi baru dengan memodifikasi dari solusi awal. Maka dari itu temperatur awal sebagai parameter dalam membentuk solusi baru dan untuk menentukan solusi terbaik dengan perhitungan probabilitas.

CENTRAL LIBRARY OF MAULANA MALIK IBRAHIM STATE ISLAMIC UNIVERSITY OF MALANG



Gambar 3.8 Flowchart *simulated annealing*

CENTRAL LIBRARY OF MAULANA MALIK IBRAHIM STATE ISLAMIC UNIVERSITY OF MALANG

3.2.6 Pembangkitan *data test*

Pada algoritma *simulated annealing* untuk mendeskripsikan solusi yang dihasilkan baik atau jelek, digunakan nilai objektif. Masing-masing solusi dihitung nilai objektifnya. Nilai objektif merupakan keluaran dari fungsi objektif. Nilai objektif ini pula yang menggambarkan fitness sebuah solusi atau biasa disebut juga nilai fitness, sehingga fungsi objektif dapat disebut juga fungsi fitness.

Fungsi objektif yang digunakan pada kasus pembangkitan data uji ini adalah fungsi cost yang diusulkan oleh (Mandyartha, 2017). Fungsi cost merupakan fungsi pinalti yang merepresentasikan jarak cabang (branch distance) yang akan dihitung bila predikat cabang tersebut tidak terpenuhi. Tabel 3.1 menunjukkan fungsi cost terhadap pemenuhan ekspresi predikat dengan a, b adalah bilangan riil dan k adalah bilangan riil positif. “Predikat A AND predikat B” “(A&&B)” dan “Predikat A OR predikat B” “(A||B)” adalah ekspresi predikat operasi logika.

Tabel 3.1 Fungsi *Cost* atau *Fitness*

Ekspresi Predikat	Cost bila tidak memenuhi Ekspresi Predikat
$a \leq b$	$a - b$
$a < b$	$a - b + k$
$a = b$	$abs\ (a\text{-}b)$
$a \neq b$	$k - abs\ (a\text{-}b)$
$a \geq b$	$b - a$
$a > b$	$b - a + k$
$A\ \&\&\ B$	$max(cost(A), cost(B))$
$A\ \|\|\ B$	$min(cost(A), cost(B))$

Pada tahap ini menjelelaskan proses pembangkitan data test secara manual menggunakan algoritma *simulated annealing* berdasarkan pada gambar 3.8. Ada

CENTRAL LIBRARY OF MAULANA MALIK IBRAHIM STATE ISLAMIC UNIVERSITY OF MALANG

beberapa langkah yang harus dilakukan, berikut langkah-langkah dalam membangkitkan data test.

1. Pemilihan *path* random yang telah terbentuk pada proses pembentukan *path* pada langkah 3.3.4.

   Path yang dipilih adalah *path* 1 = 0 – 1 – 13.

2. Menetapkan nilai suhu awal (*T*) = 2 sebagai batas iterasi dan *r* = 0,5.

3. Menetapkan nilai solusi awal (S) sebagai nilai inputan A,B,C pada kode program *benchmark* secara random. Nilai A = 2, B = 3, C = 4.

4. Menghitung nilai *fitnes* awal atau C(S) untuk kode program pada gambar 3.8 yang telah terbentuk menjadi CFG cabang target berdasarkan nilai *cost* pada tabel 3.1.

   a. Pengujian *path* 0 – 1 – 13

   b. Ekpresi predikat yang harus di eksekusi A>0 && B>0 && C>0, di invers menjadi A≤0 || B≤0 || C≤0 karena harus melalui *node* 1 yang bernilai false.

   c. – Kondisi A≤0, A bernilai 2, maka 2≤0 bernilai *false*, sehingga dihitung fungsi *cost*nya sebagai 2 – 0 = 2.

      - Kondisi B≤0, A bernilai 3, maka 3≤0 bernilai *false*, sehingga dihitung fungsi *cost*nya sebagai 3 – 0 = 3.

      - Kondisi C≤0, A bernilai 4, maka 4≤0 bernilai *false*, sehingga dihitung fungsi *cost*nya sebagai 4 – 0 = 4.

   d. Total nilai *fitness* awal atau C(S) pada Ekpresi predikat A || B || C sebagai min(2,3,4) adalah 2.

5. Menetapkan nilai solusi baru (S`) dengan nilai A = 5, B = 7, C = 6.

CENTRAL LIBRARY OF MAULANA MALIK IBRAHIM STATE ISLAMIC UNIVERSITY OF MALANG

6. Menghitung nilai *fitnes* baru atau C(S`).

   a. Pengujian *path* 0 – 1 – 13

   b. Ekpresi predikat yang harus di eksekusi A>0 && B>0 && C>0, di invers menjadi A≤0 || B≤0 || C≤0 karena harus melalui *node* 1 yang bernilai false.

   c. – Kondisi A≤0, A bernilai 5, maka 5≤0 bernilai *false*, sehingga dihitung fungsi *cost*nya sebagai 5 – 0 = 5.

      - Kondisi B≤0, A bernilai 7, maka 7≤0 bernilai *false*, sehingga dihitung fungsi *cost*nya sebagai 7 – 0 = 7.

      - Kondisi C≤0, A bernilai 5, maka 5≤0 bernilai *false*, sehingga dihitung fungsi *cost*nya sebagai 6 – 0 = 6.

   Total nilai *fitness* baru atau C(S`) pada Ekpresi predikat A || B || C sebagai min(5,7,6) adalah 5.

7. Menghitung *delta energy* ΔC nilai *fitness* baru dikurangi nilai *fitness* awal adalah 3.

8. Cek kondisi ΔC ≤ 0, 3 ≤ 0, karena bernilai *false*, maka dilakukan perhitungan *probabilitas* dengan persamaan $p = e^{(\Delta C/T)}$.

9. $p = e^{(3/2)} = 4{,}48$.

10. Cek kondisi $p > r$. Karena $p > r$, maka solusi baru (S`) ditetapkan sebagai solusi terbaik(S*).

11. Cek kondisi solusi terbaik (S*), apakah memenuhi kriteria atau tidak, jika belom memenuhi, ulangi dari langkah no 5, lakukan iterasi dengan menurunkan temperatur jika sudah memenuhi kriteria.

12. Iterasi penurunan temperatur, $Ti = \alpha * 2$ yaitu 0.9 * 2 = 1,8.

CENTRAL LIBRARY OF MAULANA MALIK IBRAHIM STATE ISLAMIC UNIVERSITY OF MALANG

Untuk perhitungan iterasi 2 sampai selesai juga menggunakan perhitungan yang sama seperti iterasi 1 sesuai dengan penetapan nilai random solusi awal.

Tabel 3.2 Hasil iterasi pembangkitan *data test*

Iterasi	Temperatur Ti = 0,9 * T	Solusi awal (S)			Nilai fitnes solusi awal C(S)	Solusi baru (S`)			Nilai fitnes solusi baru C(S`)	*delta energy* ΔC = C(S') - S(S)	ΔC ≤ 0 ?	Probabilitas p = e (ΔC/T)	p > r ? (r = 0.5)	Solusi terbaik (S*)		
		A	B	C		A	B	C						A	B	C
1	2	2	3	4	2	5	7	6	5	3	FALSE	1,64	TRUE	5	7	6
2	1,8	5	7	6	5	4	1	3	1	-4	TRUE	-	-	4	1	3
3	1,62	4	1	3	1	16	10	8	8	7	FALSE	75,26	TRUE	16	10	8
4	1,44	16	10	8	8	9	3	3	3	-5	TRUE	-	-	9	3	3
5	1,26	9	3	3	3	6	5	4	4	1	FALSE	2,21	TRUE	6	5	4
6	1,08	6	5	4	4	8	6	9	6	2	FALSE	6,37	TRUE	8	6	9
7	0,97	8	6	9	6	10	12	12	10	4	FALSE	61,78	TRUE	10	12	12
8	0,81	10	12	12	10	1	4	3	1	-9	TRUE	-	-	1	4	3
9	0,72	1	4	3	1	0	2	1	0	-1	TRUE	-	-	0	2	1

Dari perhitungan pada tabel 3.2, dapat dilihat bahwa hasil iterasi pengujian *data test* pada *path* 0 – 1 – 13 *data test* atau solusi terbaik adalah a = 0, b = 2 c = 1 dengan nilai *fitnes* bernilai 0. Pada aplikasi ini, pemberhentian proses *simulated annealing* menggunakan 2 parameter. Parameter pertama adalah jika nilai *fitnes* solusi bernilai 0. Parameter kedua adalah jika nilai solusi baru yang terbentuk bernilai sama sampai beberapa kemunculan.

3.3 Pengujian Algoritma *Simulated Annealing* (SA) Secara Otamatis

Proses pembangkitan *data test* secara otomatis menggunakan *Simulated Annealing* ini dilakukan ketika hasil *path* (jalur) dari proses *control flow graph* (CFG) telah berhasil terbentuk. Kemudian metode *Particle Swarm Optimazation* (*PSO*) ini bekerja dengan menguji masing-masing *path* atau jalur, untuk mengetahui *path* atau jalur mana yang dilalui data *test*. Berikut *source code* penerapan metode *Simulated Annealing* (SA).

CENTRAL LIBRARY OF MAULANA MALIK IBRAHIM STATE ISLAMIC UNIVERSITY OF MALANG

```
public void SimulatedAnnealing() {
        double temp = 100000;
        double coolingRate = 0.003;
        Path currentSolution = new Path();
        currentSolution.generateIndividual();
        text_output.append("\n");
        text_output.append("Total distance of initial 
solution: " + 
currentSolution.getTotalDistance()+"\n");
        Path best = new 
Path(currentSolution.getTour());
        while (temp > 1) {
            Path newSolution = new 
Path(currentSolution.getTour());
            int tourPos1 = Utility.randomInt(0 , 
newSolution.tourSize());
            int tourPos2 = Utility.randomInt(0 , 
newSolution.tourSize());
      while(tourPos1 == tourPos2) {tourPos2 = 
Utility.randomInt(0 , newSolution.tourSize());}
            gui.Node positionSwap1 = 
newSolution.getPosition(tourPos1);
            gui.Node positionSwap2 = 
newSolution.getPosition(tourPos2);
            newSolution.setPosition(tourPos2, 
positionSwap1);
            newSolution.setPosition(tourPos1, 
positionSwap2);
            int currentDistance    = 
currentSolution.getTotalDistance();
            int neighbourDistance = 
newSolution.getTotalDistance();
            double rand = Utility.randomDouble();
            if 
(Utility.acceptanceProbability(currentDistance, 
neighbourDistance, temp) > rand) {
                currentSolution = new 
Path(newSolution.getTour());
            }
            if (currentSolution.getTotalDistance() < 
best.getTotalDistance()) {
                best = new 
Path(currentSolution.getTour());
            text_output.append(best + "\n");
            }
            temp *= 1 - coolingRate;
        }
        text_output.append("========================" 
+ "\n");
        text_output.append("Final solution distance : 
" + best.getTotalDistance() + "\n");
    }
```

Gambar 3.9 *source code* proses *Simulated Annealing*

CENTRAL LIBRARY OF MAULANA MALIK IBRAHIM STATE ISLAMIC UNIVERSITY OF MALANG

3.4 Spesifikasi Sistem

Dibutuhkan beberapa platform yang digunakan untuk mengimplementasikan pencarian rute tercepat ini. Platform yang dibutuhkan berupa *software* dan *hardware*. Berikut platform yang digunakan:

a. *Software*

- Windows 10
- IDE Netbeans with Java Programming

b. *Hardware*

- Laptop *Processor AMD A10* 64 bit
- Memory 4 Gb
- Harddisk 1 TB

3.4 Evaluasi dan Validasi hasil

Pada tahap ini akan dilakukan proses evaluasi perbandingan rata-rata coverage. Evaluasi ini dilakukan dari hasil pengujian metode *simulated annealing*. Dari evaluasi tersebut kemudian dilakukan validasi hasil dengan menguji *data test* yang telah dibangkitkan pada kode program unit (*benchmark*).

CENTRAL LIBRARY OF MAULANA MALIK IBRAHIM STATE ISLAMIC UNIVERSITY OF MALANG

BAB IV

HASIL DAN PEMBAHASAN

Dalam bab ini akan membahas tentang hasil uji dan pembahasan bagaimana proses pembuatan aplikasi, uji coba dilakukan untuk mengetahui apakah aplikasi dapat berjalan sesuai dengan yang diharapkan.

4.1 Deskripsi Program

Aplikasi pembangkitan data tes otomatis kali ini dibangun dengan menggunakan bahasa pemrograman Java dan berbasis dekstop. Pembuatan aplikasi ini memanfaatkan *NetBeans* sebagai IDE (*Integrated Development Environment*). Untuk memudahkan pengguna dalam mengoperasikan aplikasi ini maka dibuat sebuah tampilan *interface* berbasi GUI (*Graphic User Interface*). Berikut ini adalah tampilan utama dari aplikasi ini.

Pada aplikasi ini juga terdapat fitur pengisian data. Data tersebut sebagai masukan (*input*). Masukan (*input*) berupa fungsi dari program unit (*source code*) yang diproses kemudian ditampilkan. Berikut tampilan *interface* dari fitur yang sudah dijelaskan di atas.

4.2 Hasil Pengujian

Pengujian merupakan bagian yang terpenting dalam proses pembuatan perangkat lunak. Pengujian ini dilakukan untuk menjamin kualitas perangkat lunak yang dibangun dan mengetahui kelemahan dari perangkat lunak yang dibangun. Kasus uji yang baik adalah yang memiliki tingkat kemungkinan tinggi untuk menemukan kerusakan yang belum ditentukan.

CENTRAL LIBRARY OF MAULANA MALIK IBRAHIM STATE ISLAMIC UNIVERSITY OF MALANG

4.2.1. Data Uji Coba

Ada tiga data uji atau *benchmark* data program publik yang akan dilakukan dalam pengujian yang akan dibuat. Penggunaan *benchmark* data program publik memungkinkan orang lain dapat memvalidasi hasil penelitian yang dilakukan. Pada pengujian ini menggunakan tiga *benchmark* data program, yaitu *Myers Triangle, Michael Triangle, Sthamer Triangle.*

a. *Myers Triangle*

Program ini menggolongkan segitiga atas dasar sisi input sebagai non-segitiga atau segitiga, yaitu sama kaki, sama sisi atau sisi tak sama panjang. Diberikan tiga input nilai di dalam parameter yang semuanya mewakili sisi segitiga.

b. *Michael Triangle*

Program ini juga menggolongkan segitiga atas dasar sisi input sebagai non-segitiga atau segitiga, yaitu sama kaki, sama sisi atau sisi tak sama panjang. Dibutuhkan tiga input nilai dengan semua dari ketiganya mewakili sisi segitiga tetapi dengan kondisi predikat yang berbeda.

c. *Sthamer Triangle*

Seperti *benchmark* program sebelumnya, program ini menggolongkan segitiga atas dasar sisi input sebagai non-segitiga atau segitiga, yaitu sama kaki, sama sisi, segitiga sudut kanan atau sisi tak sama panjang.

CENTRAL LIBRARY OF MAULANA MALIK IBRAHIM STATE ISLAMIC UNIVERSITY OF MALANG

4.2.2. Proses Pembuatan Aplikasi

Ada beberapa tahapan dalam pembuatan aplikasi pembangkitan data tes secara otomatis menggunakan algoritma *Simulated Annealing* yang akan dibuat, berikut merupakan tahapan-tahapan proses yang ada dalam pembuatan aplikasi :

a. *Parsing* Program

Pada tahapan *parsing* program, proses yang terjadi adalah tiga data uji atau *benchmark* program di *parsing* dimana *sintaks-sintaks* dari *query* akan dicek untuk menentukan agar *query* tersebut sudah sesuai dengan aturan-aturan *sintaks* (aturan grammar) dari bahasa *query*.

b. Konversi ke CFG (*Control Flow Graph*)

Pada tahapan konversi ke CFG (*Control Flow Graph*), inputan berupa data uji atau *benchmark* data program dianalisis dan diproses ke dalam bentuk *Graph* sehingga menggambarkan aliran kode program, awal program hingga akhir program.

Setelah melalui proses analisis *Control Flow Graph* (CFG) yang menghasilkan gambaran aliran *path* kode program. Berikut merupakan semua kemungkinan jalur (*path*) yang dapat dijadikan sebagai dasar dalam pembangkitan data uji, kemungkinan jalur (*path*) dipresentasikan dalam bentuk urutan huruf node.

CENTRAL LIBRARY OF MAULANA MALIK IBRAHIM STATE ISLAMIC UNIVERSITY OF MALANG

c. Penerapan Metode *Simulated Annealing* (SA)

Setelah pembangkitan data tes yang melalui proses analisis *Control Flow Graph* (CFG) selesai, pembangkitan data tes akan dilanjutkan menggunakan algoritma *simulated annealing*. *Simulated annealing* ini akan bertugas untuk proses menghasilkan data tes berdasarkan inputan dari hasil *Control Flow Graph* (CFG).

Pada tahapan penerapan metode *simulated annealing* (SA), proses ini dilakukan ketika hasil *path* atau jalur dari proses CFG telah berhasil terbentuk. Kemudian metode *simulated annealing* ini bekerja dengan menguji masing-masing *path* atau jalur, untuk mengetahui *path* atau jalur mana yang dilalui data tes.

```
public void SimulatedAnnealing() {
        double temp = 100000;
        double coolingRate = 0.003;
        Path currentSolution = new Path();
        currentSolution.generateIndividual();
        text_output.append("\n");
        text_output.append("Total distance of initial
solution: " +
currentSolution.getTotalDistance()+"\n");
        Path best = new
Path(currentSolution.getTour());
        while (temp > 1) {
            Path newSolution = new
Path(currentSolution.getTour());
            int tourPos1 = Utility.randomInt(0 ,
newSolution.tourSize());
            int tourPos2 = Utility.randomInt(0 ,
newSolution.tourSize());
      while(tourPos1 == tourPos2) {tourPos2 =
Utility.randomInt(0 , newSolution.tourSize());}
            gui.Node positionSwap1 =
newSolution.getPosition(tourPos1);
            gui.Node positionSwap2 =
newSolution.getPosition(tourPos2);
            newSolution.setPosition(tourPos2,
positionSwap1);
            newSolution.setPosition(tourPos1,
positionSwap2);
```

CENTRAL LIBRARY OF MAULANA MALIK IBRAHIM STATE ISLAMIC UNIVERSITY OF MALANG

```
int currentDistance   =
currentSolution.getTotalDistance();
            int neighbourDistance =
newSolution.getTotalDistance();
            double rand = Utility.randomDouble();
            if
(Utility.acceptanceProbability(currentDistance,
neighbourDistance, temp) > rand) {
                currentSolution = new
Path(newSolution.getTour());
            }
            if (currentSolution.getTotalDistance() <
best.getTotalDistance()) {
                best = new
Path(currentSolution.getTour());
            text_output.append(best + "\n");
            }
            temp *= 1 - coolingRate;
        }
        text_output.append("========================"
+ "\n");
        text_output.append("Final solution distance :
" + best.getTotalDistance() + "\n");
    }
```

Gambar 4.1 *Source Code* Implementasi Metode SA

4.2.3. Pengujian Aplikasi

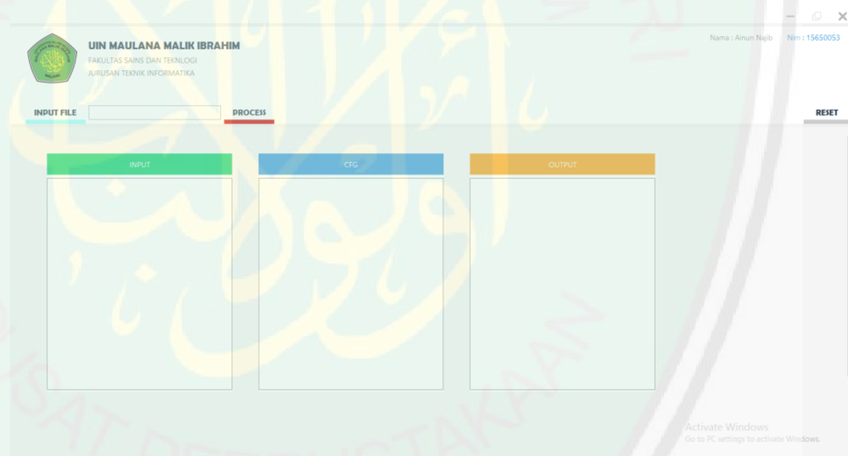
Aplikasi pembangkit data tes otomatis ini dibangun dengan menggunakan bahasa pemrograman Java. Pembuatan aplikasi ini memanfaatkan Netbeans IDE 8.0 sebagai text editor untuk penulisan kode program. Untuk mempermudah pengguna dalam mengoperasikan aplikasi ini, sehingga dibuat sebuah tampilan (*interface*) berbasi GUI (*Grapic User Intreface*) yang kemudian akan dijelaskan lebih rinci.dibawah ini.

Aplikasi pembangkit data tes otomatis ini memiliki *interface* ektraksi file *source code* (java) sebagai *input* (masukan) data awal.

CENTRAL LIBRARY OF MAULANA MALIK IBRAHIM STATE ISLAMIC UNIVERSITY OF MALANG

Gambar 4.2 Tampilan Awal Program

Pada gambar 4.2 adalah tampilan awal ketika program pertama kali dijalankan.



Gambar 4.3 Tampilan Halaman Utama

Pada gambar 4.3 merupakan halaman utama yang menampilkan 3 kolom utama yaitu, kolom *Input* yang berfungsi untuk memasukkan data uji atau sebagai *input* atau masukan awal program yang ditunjukkan pada gambar 4.4. Kolom *Control Flow Graph* (CFG) sebagai Proses yang akan menampilkan hasil *path*

CENTRAL LIBRARY OF MAULANA MALIK IBRAHIM STATE ISLAMIC UNIVERSITY OF MALANG

dan proses dari algoritma *Simulated Annealing*, yang ditunjukkan pada gambar 4.5. Kolom *Output* sebagai keluaran atau hasil dari proses data uji yang telah diuji dengan algoritma *Simulated Annealing* yang ditunjukkan pada gambar 4.6. Berikut urutan langkah dalam proses pembangkitan data tes secara otomatis, yaitu:

1. Pengambilan data uji sebagai *inputan*

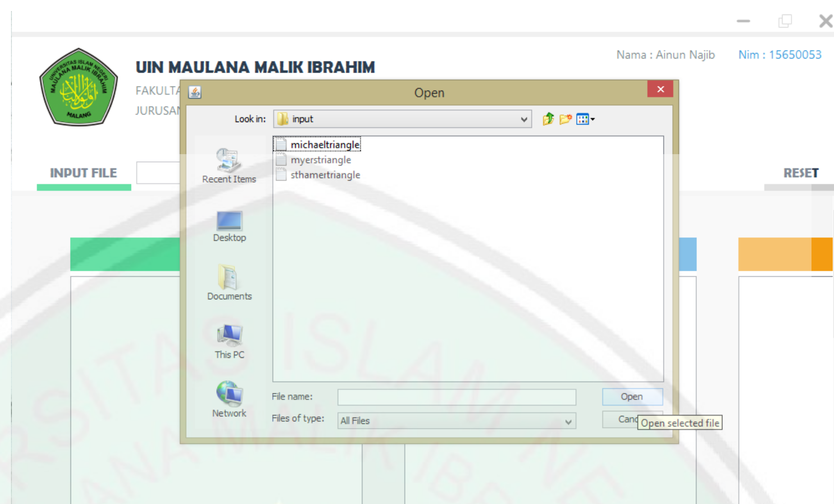
Pengambilan data uji ini dilakukan dengan mencari *file input* yang berbentuk *source code* berekstensi java sebagai masukan awal yang akan ditampilkan pada kolom Input.

Gambar 4.4 Tampilan Kolom Input

Kemudian klik "*input file*" untuk mengambil *file* data uji berupa *source code* dengan format text atau java, setelah itu akan muncul beberapa data uji pada *direktori* komputer. Pada pengujian

CENTRAL LIBRARY OF MAULANA MALIK IBRAHIM STATE ISLAMIC UNIVERSITY OF MALANG

ini menggunakan data uji *Myers Triangle*. Terlihat pada gambar berikut ini.

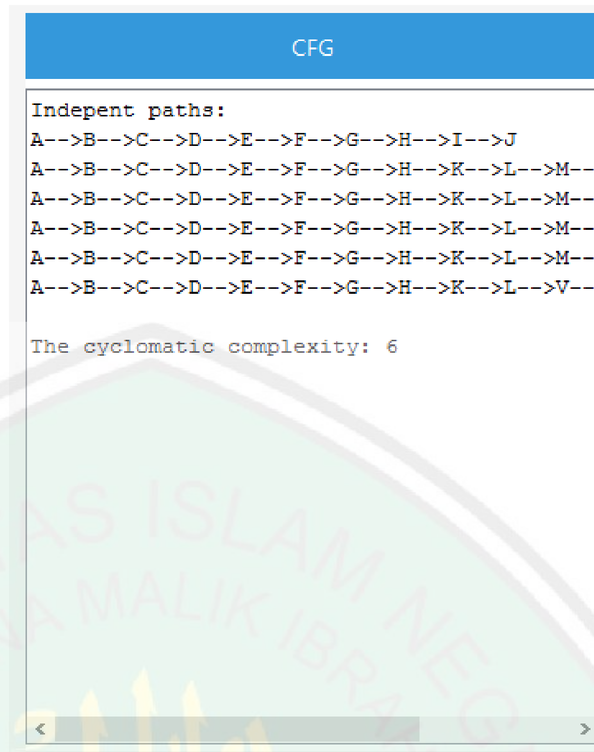


Gambar 4.5 *Browsing File* Java

2. Proses Metode Algoritma Simulated Annealing

Masukan (*input*) data uji dianalis menggunakan *control flow graph* (CFG) sehingga terbentuk *path* (jalur) seperti pada gambar 4.6. Setelah aliran (*path*) kode program terbentuk kemudian dibangkitkan data uji menggunakan metode algoritma *simulated annealing*.

CENTRAL LIBRARY OF MAULANA MALIK IBRAHIM STATE ISLAMIC UNIVERSITY OF MALANG



Gambar 4.6 Tampilan Kolom Proses

3. Hasil Output Data Uji

Masing-masing *path* yang telah terbentuk pada proses CFG dieksekusi menggunkan *simulated annealing*, sehingga menghasilkan data *test* (data uji) yang terbaik dan jalur (*path*) manakah yang benar dilalui oleh *simulated annealing*.

CENTRAL LIBRARY OF MAULANA MALIK IBRAHIM STATE ISLAMIC UNIVERSITY OF MALANG

```
OUTPUT
-------------------VALUE----------------
x : 39, y : 17, z : 4
x : 38, y : 32, z : 23
x : 22, y : 37, z : 0
-------------------OUTPUT---------------

Total distance of initial solution: 136
9 -> 26 -> 31 -> 20 -> 23 -> 1 -> 22 -> 1
12 -> 23 -> 31 -> 20 -> 9 -> 4 -> 22 -> 2
20 -> 23 -> 21 -> 11 -> 25 -> 27 -> 31 ->
21 -> 9 -> 11 -> 18 -> 15 -> 20 -> 4 -> 1
32 -> 22 -> 23 -> 11 -> 18 -> 4 -> 1 -> 9
11 -> 4 -> 1 -> 12 -> 9 -> 22 -> 31 -> 30
18 -> 23 -> 22 -> 31 -> 26 -> 30 -> 25 ->
27 -> 26 -> 31 -> 20 -> 32 -> 30 -> 22 ->
30 -> 31 -> 32 -> 27 -> 26 -> 20 -> 22 ->
27 -> 32 -> 31 -> 30 -> 25 -> 20 -> 21 ->
27 -> 31 -> 32 -> 30 -> 25 -> 20 -> 21 ->
23 -> 27 -> 30 -> 32 -> 31 -> 25 -> 26 ->
========================
Final solution distance : 54
```

Gambar 4.7 Hasil Output Proses Implementasi Metode *Simulated Annealing*

Pada gambar 4.7 merupakan hasil data *test* dari *myers triangle* dan jarak *path* yang dilalui setelah diproses oleh *simulated annealing* (SA).

4.3 Pembahasan

Setelah pembuatan program untuk metode yang diusulkan sudah dibuat, selanjutnya adalah melakukan pengujian otomatis. Pengujian akan dilakukan dengan masing-masing data uji *benchmark* program yang digunakan. Dari beberapa percobaan didapatkan pengaturan yang cukup memadai untuk digunakan didalam pengujian.

4.3.1. Hasil Uji Coba

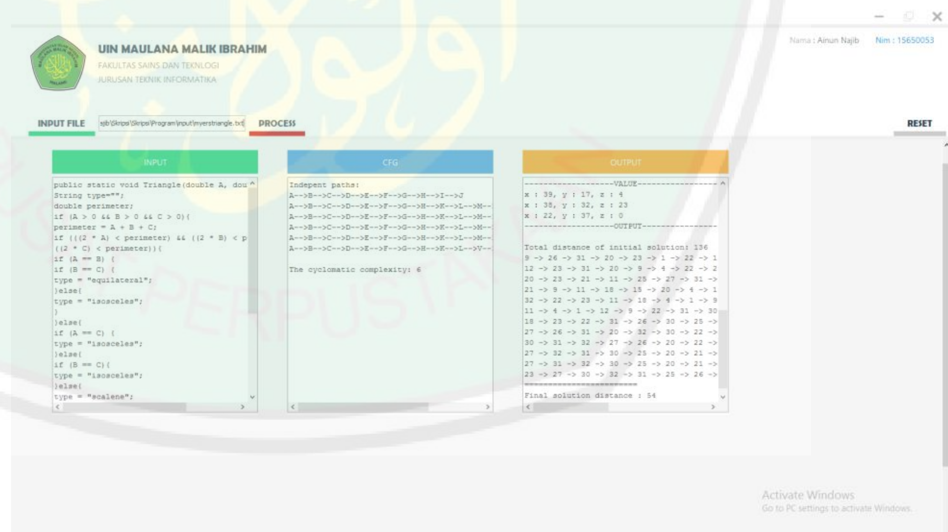
Setelah pembutan program untuk metode yang diusulkan sudah dibuat, selanjutnya adalah melakukan pengujian pada ketiga *benchmark* program

CENTRAL LIBRARY OF MAULANA MALIK IBRAHIM STATE ISLAMIC UNIVERSITY OF MALANG

yakni, menggunakan data uji dari *myers triangle*, *michael triangle*, dan *sthamer triangle*. Pengujian akan dilakukan dengan masing-masing *benchmark* program yang digunakan. Penentuan parameter dan operator berdasarkan hasil beberapa pengaturan yang dilakukan oleh peneliti. Dari beberapa percobaan didapatkan beberapa pengaturan yang cukup memadai untuk digunakan di dalam pengujian.

4.3.1.1 Pengujian untuk *Benchmark* Program *Myers Triangle*

Pengujian yang dilakukan bertujuan untuk mengetahui *path* atau jalur yang terbentuk pada *Control Flow Graph* (CFG) dan hasil pembangkitan data tes dari *path* yang diproses dengan *simulated annealing*. *Control Flow Graph* yang dihasilkan memiliki 14 *node* dengan predikat *node* berjumlah 6. Sedangkan jalur independen atau *path* yang terbentuk yakni sebanyak 6 *path*.



Gambar 4.8 Hasil Analisa *Myers Triangle*

Pada gambar 4.8, aplikasi menampilkan hasil analisa jarak *path* dan hasil akhir yang paling optimal berdasarkan metode *Simulated*

CENTRAL LIBRARY OF MAULANA MALIK IBRAHIM STATE ISLAMIC UNIVERSITY OF MALANG

Annealing (SA). Kolom *control flow graph* (CFG) menunjukkan hasil *path* untuk mendapatkan pemecahan masalah yang optimal. Berikut semua kemungkinan jalur (*path*) yang dihasilkan dan akan dijadikan dasar dalam pembangkita data uji.

Indepent paths:

A-->B-->C-->D-->E-->F-->G-->H-->I-->J

A-->B-->C-->D-->E-->F-->G-->H-->K-->L-->M-->N-->O-->P-->Q-->R

A-->B-->C-->D-->E-->F-->G-->H-->K-->L-->M-->N-->O-->P-->S

A-->B-->C-->D-->E-->F-->G-->H-->K-->L-->M-->T-->U

A-->B-->C-->D-->E-->F-->G-->H-->K-->L-->M-->X

A-->B-->C-->D-->E-->F-->G-->H-->K-->L-->V-->W-->X

The cyclomatic complexity: 6

Kolom ouput menampilkan hasil dari nilai data tes yang paling konvergen untuk pemecahan masalah *myers triangle*.

Tabel 4.1 menunjukkan hasil pengujian data *test* pada *benchmark myers triangle*.

Tabel 4.1 Hasil data *test myers triangle*

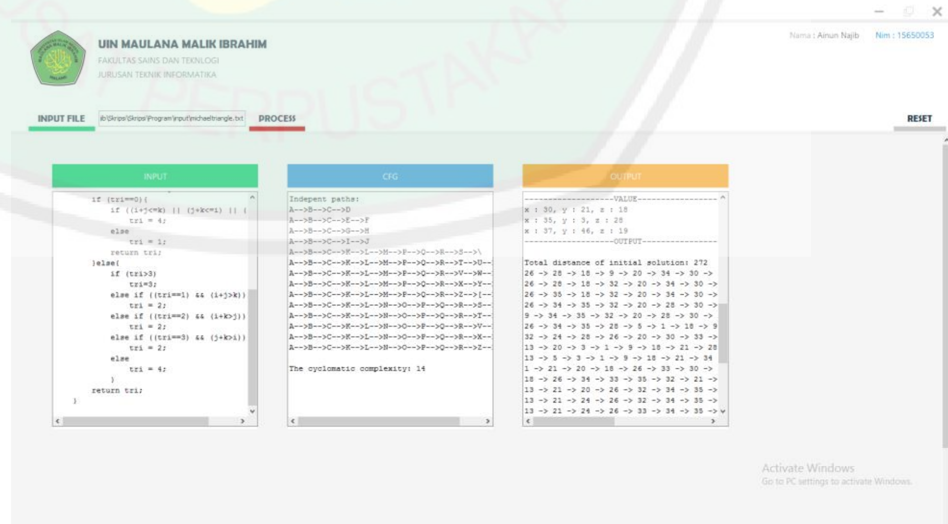
Hasil Data *Test*			Validasi Hasil
X	Y	Z	
39	17	4	Bukan Segitiga
38	32	23	Segitiga Sembarang
22	37	0	Bukan Segitiga

CENTRAL LIBRARY OF MAULANA MALIK IBRAHIM STATE ISLAMIC UNIVERSITY OF MALANG

Pada tabel 4.1 menampilkan beberapa hasil *data test* yang berhasil di*generate* menggunakan metode *simulated annealing*. *Data test* tersebut dihasilkan dari proses *simulated annealing* dengan menurunkan temperatur, dimana fungsi dari temperatur adalah suatu nilai kendali yang membuat suatu *node* acak dapat bergerak naik atau tidak. *Data test* x = 38, y = 32, z = 23 merupakan hasil *data test* terbaik pada *benchmark myers triangle* yang berhasil di*generate*, dimana setelah uji validasi *data test* tersebut menunjukan hasil yang di cari yaitu tipe segitiga.

4.3.1.2 Pengujian untuk *Benchmark* Program *Michael Triangle*

Pengujian yang dilakukan bertujuan untuk mengetahui *path* atau jalur yang terbentuk pada *Control Flow Graph* (CFG) dan hasil pembangkitan data tes dari *path* yang diproses dengan *simulated annealing*. *Control Flow Graph* yang dihasilkan memiliki 26 node dengan predikat *node* berjumlah 11. Sedangkan jalur independen atau *path* yang terbentuk yakni sebanyak 14 *path*.



Gambar 4.9 Hasil Analisa *Michael Triangle*

CENTRAL LIBRARY OF MAULANA MALIK IBRAHIM STATE ISLAMIC UNIVERSITY OF MALANG

Pada gambar 4.9, aplikasi menampilkan hasil analisa jarak *path* dan hasil akhir yang paling optimal berdasarkan metode *Simulated Annealing* (SA). Kolom *control flow graph* (CFG) menunjukkan hasil *path* untuk mendapatkan pemecahan masalah yang optimal. Berikut semua kemungkinan jalur (*path*) yang dihasilkan dan akan dijadikan dasar dalam pembangkita data uji.

Indepent paths:

A-->B-->C-->D

A-->B-->C-->E-->F

A-->B-->C-->G-->H

A-->B-->C-->I-->J

A-->B-->C-->K-->L-->M-->P-->Q-->R-->S

A-->B-->C-->K-->L-->M-->P-->Q-->R-->T-->U

A-->B-->C-->K-->L-->M-->P-->Q-->R-->V-->W

A-->B-->C-->K-->L-->M-->P-->Q-->R-->X-->Y

A-->B-->C-->K-->L-->M-->P-->Q-->R-->Z

A-->B-->C-->K-->L-->N-->O-->P-->Q-->R-->S

A-->B-->C-->K-->L-->N-->O-->P-->Q-->R-->T-->U

A-->B-->C-->K-->L-->N-->O-->P-->Q-->R-->V-->W

A-->B-->C-->K-->L-->N-->O-->P-->Q-->R-->X-->Y

A-->B-->C-->K-->L-->N-->O-->P-->Q-->R-->Z

The cyclomatic complexity: 14

Kolom ouput menampilkan hasil dari nilai data tes yang paling konvergen untuk pemecahan masalah *michael triangle*.

CENTRAL LIBRARY OF MAULANA MALIK IBRAHIM STATE ISLAMIC UNIVERSITY OF MALANG

Tabel 4.2 menunjukkan hasil pengujian data *test* pada *benchmark michael triangle*.

Tabel 4.2 Hasil data *test michael triangle*

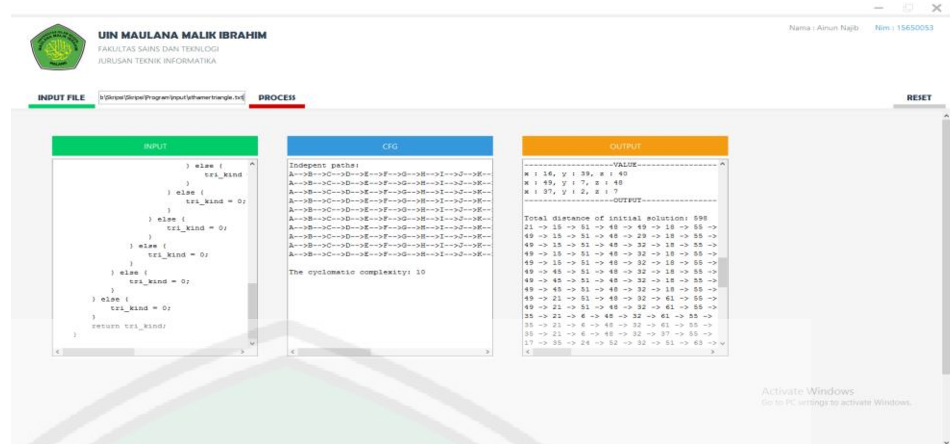
Hasil Data *Test*			Validasi Hasil
X	Y	Z	
30	21	18	Segitiga
35	3	28	Bukan Segitiga
37	46	19	Segitiga

Pada tabel 4.2 menampilkan beberapa hasil *data test* yang berhasil di*generate* menggunakan metode *simulated annealing*. *Data test* tersebut dihasilkan dari proses *simulated annealing* dengan menurunkan temperatur, dimana fungsi dari temperatur adalah suatu nilai kendali yang membuat suatu *node* acak dapat bergerak naik atau tidak. *Data test* x = 30, y = 21, z = 18 merupakan hasil *data test* terbaik pada *benchmark michael triangle* yang berhasil di*generate*, dimana setelah uji validasi *data test* tersebut menunjukan hasil yang di cari yaitu tipe segitiga.

4.3.1.3 Pengujian untuk *Benchmark* Program *Sthamer Triangle*

Pengujian yang dilakukan bertujuan untuk mengetahui *path* atau jalur yang terbentuk pada *Control Flow Graph* (CFG) dan hasil pembangkitan data tes dari *path* yang diproses dengan *simulated annealing*. *Control Flow Graph* yang dihasilkan memiliki 29 node dengan predikat *node* berjumlah 13. Sedangkan jalur independen atau *path* yang terbentuk yakni sebanyak 10 *path*.

CENTRAL LIBRARY OF MAULANA MALIK IBRAHIM STATE ISLAMIC UNIVERSITY OF MALANG



Gambar 4.10 Hasil Analisa *Sthamer Triangle*

Pada gambar 4.10, aplikasi menampilkan hasil analisa jarak *path* dan hasil akhir yang paling optimal berdasarkan metode *Simulated Annealing* (SA). Kolom *control flow graph* (CFG) menunjukkan hasil *path* untuk mendapatkan pemecahan masalah yang optimal.

Berikut semua kemungkinan jalur (*path*) yang dihasilkan dan akan dijadikan dasar dalam pembangkita data uji.

Indepent paths:

A-->B-->C-->D-->E-->F-->G-->H-->I-->J-->K-->L-->M-->N

A-->B-->C-->D-->E-->F-->G-->H-->I-->J-->K-->L-->M-->P-->Q-->R-->S-->T-->U-->V-->W-->X-->Y-->Z

A-->B-->C-->D-->E-->F-->G-->H-->I-->J-->K-->L-->O-->P-->Q-->R-->S-->T-->U-->V-->W-->X-->Y-->Z

A-->B-->C-->D-->E-->F-->G-->H-->I-->J-->K-->L-->O-->P-->Q-->R-->S-->T-->U-->V-->W-->X-->Y

A-->B-->C-->D-->E-->F-->G-->H-->I-->J-->K-->L-->O-->P-->Q-->R-->S-->T-->U-->V-->a-->b

CENTRAL LIBRARY OF MAULANA MALIK IBRAHIM STATE ISLAMIC UNIVERSITY OF MALANG

A-->B-->C-->D-->E-->F-->G-->H-->I-->J-->K-->L-->O-->P-->Q-->R-->S-->T-->U-->V-->c-->d

A-->B-->C-->D-->E-->F-->G-->H-->I-->J-->K-->L-->O-->P-->Q-->R-->S-->T-->U-->V-->e-->f

A-->B-->C-->D-->E-->F-->G-->H-->I-->J-->K-->L-->O-->P-->Q-->R-->S-->T-->U-->V-->g-->h

A-->B-->C-->D-->E-->F-->G-->H-->I-->J-->K-->L-->O-->P-->Q-->R-->S-->T-->U-->V-->i-->j

A-->B-->C-->D-->E-->F-->G-->H-->I-->J-->K-->L-->O-->P-->Q-->R-->S-->T-->U-->V-->k-->l

The cyclomatic complexity: 10

Kolom ouput menampilkan hasil dari nilai data tes yang paling konvergen untuk pemecahan masalah *sthamer triangle*.

Tabel 4.3 menunjukkan hasil pengujian data *test* pada *benchmark sthamer triangle*.

Tabel 4.3 Hasil data *test sthamer triangle*

Hasil Data *Test*			Validasi Hasil
X	Y	Z	
16	39	40	Segitiga
49	7	48	Segitiga
37	2	7	Bukan Segitiga

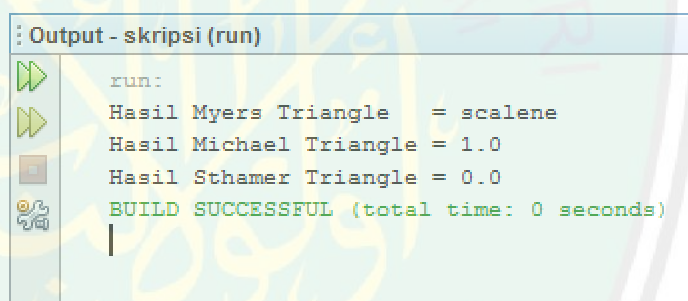
Pada tabel 4.3 menampilkan beberapa hasil *data test* yang berhasil di*generate* menggunakan metode *simulated annealing*. *Data test* tersebut

CENTRAL LIBRARY OF MAULANA MALIK IBRAHIM STATE ISLAMIC UNIVERSITY OF MALANG

dihasilkan dari proses *simulated annealing* dengan menurunkan temperatur, dimana fungsi dari temperatur adalah suatu nilai kendali yang membuat suatu *node* acak dapat bergerak naik atau tidak. *Data test* x = 16, y = 39, z = 40 merupakan hasil *data test* terbaik pada *benchmark michael triangle* yang berhasil di*generate*, dimana setelah uji validasi *data test* tersebut menunjukan hasil yang di cari yaitu tipe segitiga.

4.3.2 Validasi Hasil

Proses validasi dilakukan setelah hasil data *test* pengujian otomatis pada masing-masing *benchmark* data berhasil didapatkan. Data *test* tersebut kemudian di uji validasi dengan menginputkan atau sebagai *inputan* x, y, dan z terhadap kode program unit (*benchmark*).



Gambar 4.11 Hasil Validasi Data *Test*

Pada gambar 4.11 merupakan hasil uji validasi salah satu data *test* pada ketiga *benchmark*. Pada *benchmark Myers Triangle*, uji validasi dilakukan menggunakan salah satu sampel yang berhasil didapatkan pada tabel 4.1, dengan data *test* x=38, y=32, z=23 dimana menghasilkan tipe segitiga sembarang. Pada *benchmark Michael Triangle*, uji validasi dilakukan menggunakan salah satu sampel yang berhasil didapatkan pada tabel 4.2, dengan data *test* x=30, y=21, z=18 dimana menghasilkan tipe segitiga. Sedangkan pada *benchmark*

CENTRAL LIBRARY OF MAULANA MALIK IBRAHIM STATE ISLAMIC UNIVERSITY OF MALANG

Sthamer Triangle, uji validasi dilakukan menggunakan salah satu sampel yang berhasil didapatkan pada tabel 4.3, dengan data *test* x=37, y=2, z=7 dimana menghasilkan tipe bukan segitiga.

4.4 Integrasi Sains dan Islam

Posisi Al-Qur'an terhadap ilmu pengetahuan dan teknologi dapat dijelaskan dengan jalan mencari sumber ilmu dan sumber cara mengembangkan ilmu menjadi teknologi. Pengembangan teknologi ini meliputi pengujian terhadapa *test data* yang dibuat *output* secara otomatis. Sistem ini dibuat untuk mempermudah dalam pengembangan teknologi di bidang rekayasa perangkat lunak yang bermanfaat bagi orang lain.

Hal ini dijelaskan dalam kandungan surat Ali-Imran/3:186, yaitu:

Artinya: "Kamu sungguh-sungguh akan diuji terhadap hartamu dan dirimu. Dan (juga) kamu benar-benar akan mendengar dari orang-orang yang diberi al-Kitab sebelum kamu dan dari orang-orang yang mempersekutukan Allah, gangguan yang banyak yang menyakitkan hati. Jika kamu bersabar dan bertakwa, maka sesungguhnya yang demikian itu termasuk urusan yang patut diutamakan." (QS. Ali-Imran/3 : 186).

لَتُبْلَوُنَّ فِي أَمْوَالِكُمْ وَأَنْفُسِكُمْ وَلَتَسْمَعُنَّ مِنَ الَّذِينَ أُوتُوا الْكِتَابَ مِنْ قَبْلِكُمْ وَمِنَ الَّذِينَ أَشْرَكُوا أَذًى كَثِيرًا وَإِنْ تَصْبِرُوا وَتَتَّقُوا فَإِنَّ ذَٰلِكَ مِنْ عَزْمِ الْأُمُورِ

Tafsir Ibnu Katsir menerangkan Firman Allah SWT, لَتُبْلَوُنَّ فِي أَمْوَالِكُمْ وَأَنْفُسِكُمْ *"Kamu sungguh-sungguh akan diuji terhadap hartamu dan dirimu"* ayat ini memiliki makna pada ayat yang lain, yaitu firman-Nya *"Dan sungguh akan Kami berikan cobaan kepada kalian dengan sedikit ketakutan, kelaparan, kekurangan harta, jiwa, dan buah-buahan"* (QS. Al-Baqarah 2: 155), hingga

CENTRAL LIBRARY OF MAULANA MALIK IBRAHIM STATE ISLAMIC UNIVERSITY OF MALANG

akhir ayat berikutnya. Dengan kata lain, seorang mukmin itu harus diuji terhadap sesuatu dari hartanya atau dirinya atau anaknya atau istrinya sesuai dengan tingkatan kadar agamanya, apabila agamanya kuat, maka ujiannya lebih dari yang lain. وَأَنْفُسِكُمْ وَلَتَسْمَعُنَّ مِنَ الَّذِينَ أُوتُوا الْكِتَابَ مِنْ قَبْلِكُمْ وَمِنَ الَّذِينَ أَشْرَكُوا أَذًى كَثِيرًا *"Dan (juga) kamu benar-benar akan mendengar dari orang-orang yang diberi al-Kitab sebelum kamu dan dari orang-orang yang mempersekutukan Allah, gangguan yang banyak yang menyakitkan hati"* maknanya Allah SWT berfirman kepada orang-orang mukmin ketika mereka tiba di Madinah sebelum Perang Badar untuk meringankan beban mereka dari tekanan gangguan yang menyakitkan hati yang dilakukan oleh kaum Ahli Kitab dan kaum musyrik. Sekaligus memerintahkan mereka agar bersikap pemaaf dan bersabar serta memberikan ampunan hingga Allah memberikan jalan keluar dari hal tersebut. وَإِنْ تَصْبِرُوا وَتَتَّقُوا فَإِنَّ ذَٰلِكَ مِنْ عَزْمِ الْأُمُورِ *"Jika kamu bersabar dan bertakwa, maka sesungguhnya yang demikian itu termasuk urusan yang patut diutamakan"* maknanya bahwa Nabi dan para sahabatnya di masa lalu selalu bersikap pemaaf terhadap orang-orang musyrik dan Ahli Kitab, sesuai dengan perintah Allah kepada mereka, dan mereka bersabar dalam menghadapi gangguan yang menyakitkan. Tersebutlah bahwa Rasulullah bersikap pemaaf sesuai dengan pengertiannya dari apa yang diperintahkan oleh Allah kepadanya, sehingga Allah mengizinkan kepada beliau terhadap mereka (yakni bertindak terhadap mereka).

Ujian hidup setiap makhluk-Nya bertujuan untuk mengangkat derajat mereka. Begitu pula dengan pengujian perangkat lunak, dengan membuat *data test* secara otomatis dapat menguji perangkat lunak yang dikembangkan

CENTRAL LIBRARY OF MAULANA MALIK IBRAHIM STATE ISLAMIC UNIVERSITY OF MALANG

sehingga apabila ada kekurangan dapat diperbaiki. Dengan itu perangkat lunak akan semakin berkualitas untuk kedepannya. Pengembangan ini dimaksudkan untuk kemajuan teknologi melalui pengujian otomatis pada perangkat lunak menggunakan *simulated annealing*.

CENTRAL LIBRARY OF MAULANA MALIK IBRAHIM STATE ISLAMIC UNIVERSITY OF MALANG

BAB V

PENUTUP

5.1. Kesimpulan

Berdasarkan pada hasil penelitian yang telah dilakukan, dalam skripsi ini berhasil membangkitkan aplikasi pembangkit *data test* otomatis. Kinerja metode *simulated annealing* pada aplikasi pembangkit *data test* secara otomatis berhasil menghasilkan *data test*terbaik dari masing-masing uji coba dalam waktu singkat. Pada *benchmark myers triangle* menghasilkan 6 *path* atau jalur yang terbentuk dengan *data test* x = 38, y = 32, z = 23 merupakan hasil *data test* terbaik yang berhasil di*generate*. *Benchmark myers triangle* menghasilkan 14 *path* atau jalur yang terbentuk dengan *data test* x = 30, y = 21, z = 18 merupakan hasil *data test* terbaik yang berhasil di*generate*. *Benchmark myers triangle* menghasilkan 10 *path* atau jalur yang terbentuk dengan *data test* x = 16, y = 39, z = 40 merupakan hasil *data test* terbaik yang berhasil di*generate*.

5.2. Saran

Peneliti menyadari bahwa penelitian ini masih banyak kekurangan yang diperlukan pengembangannya agar mencapai kinerja yang lebih baik lagi. Berikut beberapa saran yang dapat dijadikan masukkan untuk penelitian selanjutnya:

1. Pengujian dapat menggunakan atau mengkombinasikan dengan metode yang lain, agar menghasilkan *data test* yang lebih baik dan lengkap.
2. Pada aplikasi ini setelah *generate data test* harus di*close* terlebih dahulu sebelum men*generate* pengujian yang lain. Sehingga peneliti selanjutnya dapat memperbaiki sistem.

CENTRAL LIBRARY OF MAULANA MALIK IBRAHIM STATE ISLAMIC UNIVERSITY OF MALANG

DAFTAR PUSTAKA

Alshraideh, M., Mahafzah, B. A., & Al-Sharaeh, S. (2011). *A multiple-population genetic algorithm for branch coverage test data generation*. Software Quality Journal, 19(3), 489-513.

Díaz., Tuya., & Blanco. (2013). *Automated software testing using a metaheuristic technique based on tabu search. Autom*. Softw. Eng.

Fitriani, Raden A., & Hermadi, Irman. (2018). *Instrumentasi Kode Program Secara Otomatis untuk Path Testing*. Jurnal Ilmu Komputer, Volume 5.

Ibn Katsir, Abu al-Fida' Ismail. *Tafsir Ibnu Katsir*. Terj. M. Abdul Ghoffar E.M dan Abu Ihsan al-Atsari. Jilid VI. Cet. I. Bogor: Pustaka Imam asy-Syafi'i, 2014.

Mandyartha. (2017). *Pembangkitan Data Uji Menggunakan Algoritma Genetika Multi-populasi Fuzzy Adaptif.* Surabaya: Institut Teknologi Adhi Tama.

Manikumar, T., Kumar, Sanjeev., & Maruthamuthu, R. (2016). *Automated test data generation for branch testing using incremental genetic algorithm*. Vol. 41. pp. 959–976. Indian Academy of Sciences

Maulana, R., Romi., & Catur. (2015). *Integrasi Pareto Fitness, Multiple-Population dan Temporary Population pada Algoritma Genetika untuk Pembangkitan Data Test pada Pengujian Perangkat Lunak*. Journal of Software Engineering, Vol. 1, No. 2.

McMinn. (2004). *Search-based software test data generation: a survey*. Verif. Reliab.

CENTRAL LIBRARY OF MAULANA MALIK IBRAHIM STATE ISLAMIC UNIVERSITY OF MALANG

P. McMinn. (2011). *Search-Based Software Testing: Past, Present and Future*. 2011 IEEE Fourth Int. Conf. Softw. Testing, Verif. Valid. Work.

Pachauri,. & Srivastava. (2013). *Automated test data generation for branch testing using genetic algorithm: An improved approach using branch ordering, memory and elitism*. J. Syst. Softw., vol. 86, no. 5, pp. 1191–1208.

Patil,. & Nikumbh. (2012). *Pair-wise Testing Using Simulated Annealing*. Procedia Technol., vol. 4, pp. 778–782.

Pressman, Roger S, Ph.D. 2002. Rekayasa Perangkat Lunak : Pendekatan praktisi (Buku I). Yogyakarta: Penerbit Andi.

Rina, Violyta. (2009). *Pengujian Perangkat Lunak. Fakultas Ilmu Komputer Universitas Indonesia*.

S. Ali, L. C. Briand, H. Hemmati, and R. K. Panesar-Walawege. (2010). *A systematic review of the application and empirical investigation of search-based test case generation*. IEEE Trans. Softw. Eng., vol. 36, no. 6, pp. 742–762.

Srivastava,. & Baby. (2010). *Automated Software Testing Using Metahurestic Technique Based on an Ant Colony Optimization*. Int. Symp. Electron. Syst. Des., pp. 235–240.

Sommerville. (2003). *Software Engineering (Rekayasa Perangkat Lunak)*. Edisi 6. Jilid 2. Jakarta: Erlangga.

Suyanto. (2010). *Algoritma optimasi : deterministik atau probabilitik*. Yogyakarta : Graha Ilmu.

CENTRAL LIBRARY OF MAULANA MALIK IBRAHIM STATE ISLAMIC UNIVERSITY OF MALANG

Utting, M., & Legeard, B. (2007). *Pactical Model-based testing: A Tools Approach*. Morgan Kaufmann Publisher Inc : San Francisco.USA.

Yao,. & Gong. (2014). *Genetic Algorithm-Based Test Data Generation for Multiple Paths via Individual Sharing*," Comput. Intell. Neurosci., vol. 2014, pp. 1–12.

CENTRAL LIBRARY OF MAULANA MALIK IBRAHIM STATE ISLAMIC UNIVERSITY OF MALANG

LAMPIRAN

1. *Benchmark* program *Myers Triangle*

```
public static void Triangle(double A, double B, double C){
String type="";
double perimeter;
if (A > 0 && B > 0 && C > 0){
perimeter = A + B + C;
if (((2 * A) < perimeter) && ((2 * B) < perimeter) &&
((2 * C) < perimeter)){
if (A == B) {
if (B == C) {
type = "equilateral";
}else{
type = "isosceles";}
}else{
if (A == C) {
type = "isosceles";
}else{
if (B == C){
type = "isosceles";
}else{
type = "scalene";}
}}
}else{
type = "notriangle";}
}
else
{
type = "notriangle";}
}
```

2. *Benchmark* program *Michael Triangle*

```
int triangle(double i, double j, double k){
        int tri = 0;
        if ((i<=0) || (j<=0) || (k<=0))
            return 4;
        else if (i==j)
            tri += 1;
        else
        if (i==k)
            tri += 2;
        else
        if (j==k)
            tri += 3;
        else
        if (tri==0){
            if ((i+j<=k) || (j+k<=i) || (i+k<=j))
                tri = 4;
            else
```


CENTRAL LIBRARY OF MAULANA MALIK IBRAHIM STATE ISLAMIC UNIVERSITY OF MALANG

```
                tri = 1;
            return tri;
        }else{
            if (tri>3)
                tri=3;
            else if ((tri==1) && (i+j>k))
                tri = 2;
            else if ((tri==2) && (i+k>j))
                tri = 2;
            else if ((tri==3) && (j+k>i))
                tri = 2;
            else
                tri = 4;}
        return tri;}
```

3. *Benchmark* program *Sthamer Triangle*

```
static int triangle3(double a, double b, double c) {
        double perim;
        int tri_kind;
        if (a > 0) {
            if (b > 0) {
                if (c > 0) {
                perim = a + b + c;
                if ((2 * a) < perim) {
                if ((2 * b) < perim) {
                if ((2 * c) < perim) {
                if (a == b) {
                if (b == c) {
                   tri_kind = 2;
                   } else {
                   tri_kind = 1;}
                   } else {
                if (a == c) {
                   tri_kind = 1;
                   } else {
                if (b == c) {
                   tri_kind = 1;
                if ((a * a + b * b) == (c * c)) {
                   tri_kind = 3;
                   } else {
                if ((b * b + c * c) == (a * a)) {
                   tri_kind = 3;
                   } else {
                if ((a * a + c * c) == (b * b)) {
                   tri_kind = 3;
                   } else {
                   tri_kind = 4;}
                   }}}}
                   } else {
                   tri_kind = 0;}
                   } else {
                   tri_kind = 0;}
                   } else {
                        tri_kind = 0;
                   }
```

CENTRAL LIBRARY OF MAULANA MALIK IBRAHIM STATE ISLAMIC UNIVERSITY OF MALANG

```
                } else {
                    tri_kind = 0;}
            } else {
                tri_kind = 0;}
        } else {
            tri_kind = 0;}
        return tri_kind;
    }
```

CENTRAL LIBRARY OF MAULANA MALIK IBRAHIM STATE ISLAMIC UNIVERSITY OF MALANG