ALGORITMA *PARTICLE SWARM OPTIMIZATION* UNTUK PEMBANGKITAN DATA *TEST* SECARA OTOMATIS PADA PENGUJIAN PERANGKAT LUNAK

SKRIPSI

OLEH :
ALIFIYAH PUSAKA DEWI SAMUDRA
NIM. 15650042

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

ALGORITMA *PARTICLE SWARM OPTIMIZATION* 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 :
ALIFIYAH PUSAKA DEWI SAMUDRA
NIM. 15650042

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

LEMBAR PERSETUJUAN

ALGORITMA *PARTICLE SWARM OPTIMIZATION* UNTUK PEMBANGKITAN DATA *TEST* SECARA OTOMATIS PADA PENGUJIAN PERANGKAT LUNAK

SKRIPSI

Oleh :

ALIFIYAH PUSAKA DEWI SAMUDRA
NIM.15650042

Telah diperiksa dan disetujui untuk Diuji

Tanggal: 27 November 2020

Pembimbing I	Pembimbing II
H. Fatchurrohman, M.Kom	Ainatul Mardhiyah, M.Cs
NIP. 19700731 200501 1 002	NIDT. 19860330 20160801 2 075

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

Dr.Cahyo Crysdian
NIP.19740424 200901 1 008

LEMBAR PENGESAHAN

ALGORITMA *PARTICLE SWARM OPTIMIZATION* UNTUK PEMBANGKITAN DATA *TEST* SECARA OTOMATIS PADA PENGUJIAN PERANGKAT LUNAK

SKRIPSI

Oleh:
ALIFIYAH PUSAKA DEWI SAMUDRA
NIM. 15650042

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	Dr. Muhammad Faisal, M.T	( )
	: NIP. 19740510 200501 1 007	
2. Ketua Penguji	Ajib Hanani, M.T	( )
	: NIDT. 19840731 20160801 1 076	
3. Sekretaris Penguji	H. Fatchurrohman, M.Kom	( )
	: NIP. 19700731 200501 1 002	
4. Anggota Penguji	Ainatul Mardhiyah, M.Cs	( )
	: NIDT. 19860330 20160801 2 075	

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

Dr. Cahyo Crysdian
NIP. 19740424 200901 1 008

PERNYATAAN KEASLIAN TULISAN

Saya yang bertanda tangan dibawah ini :

Nama	: Alifiyah Pusaka Dewi Samudra
NIM	15650042
Fakultas/Jurusan	: Sains dan Teknologi/Teknik Informatika
Judul Skripsi	: Algoritma *Particle Swarm Optimization* 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, 07 Desember 2020

Yang Membuat Pernyataan

Alifiyah Pusaka Dewi S

Nim. 15650042

HALAMAN MOTTO

“ Jangan menjelaskan tentang dirimu kepada siapapun
karena yang menyukaimu tidak butuh itu dan
yang membencimu tidak percaya itu”

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 papa, mama, ayah, nenek, kakek. Sebagaimana Bapak Isbahar Buamona dan Ibu Dina Kusniana. Tidak lupa adik – adikku, tante, bude dan saudara yang lainnya 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 terutama Ainun Najib yang selalu sama saya, Reza Endah Pangestuti, Berlian Gita Cahyani, Imroatut Taslimah, Iluk Auliya, dan lain – lain.

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.

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 *Particle Swarm Optimization* 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.
3. Dr. Cahyo Crysdian selaku Ketua Jurusan Teknik Informatika Fakultas Sains dan Teknologi UIN Maulana Malik Ibrahim Malang.

4. Bapak H. Fatchurrohman, M.Kom selaku pembimbing I dan Ibu Ainatul Mardhiyah, M.Cs 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

DAFTAR ISI

DAFTRA GAMBAR

DAFTAR TABEL

ABSTRAK

Samudra, Alifiyah Pusaka Dewi. 2020. ***Algoritma Particle Swarm Optimization 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) H. Fatchurrohman, M.Kom. (II) Ainatul Mardhiyah, M.Cs.

Kata Kunci : *Particle Swarm Optimization, Data Test, Perangkat Lunak*

Teknologi dalam bidang perangkat lunak yang berkualitas telah banyak diciptakan. Namun, perkembangan kualitas perangkat lunak terkadang tidak melalui proses terpenting yaitu pengujian perangkat lunak. Prosedur pengujian sistem tidak dapat dilakukan secara sembarangan, melainkan memiliki skenario atau kerangka uji yang dibuat oleh penguji sistem. Pengujian perangkat lunak dapat dilakukan dengan pembangkitan data *test*, yaitu menggunakan data *test myers triangle dan sthamer triangle* sebagai data uji atau *inputan*, langkah pertama dengan mengubah data *test* menjadi *Control Flow Graph* (*CFG*) yang kemudian menjadi jalur *path* atau *node* setelah itu diproses menggunakan metode *Particle Swarm Optimation* untuk mencari hasil akhir atau *output* nilai dari variable A, B dan C yang menjadi nilai sisi dari segitiga. Hasil pengujian *variabel myers triangle* mendapatkan nilai *variable* terkecil dibandingkan *variable* yang lainnya, dan terbukti metode *Particle Swarm Optimation* bekerja sangat tepat dan menghasilkan *kualifikasi* hasil segitiga tak sama panjang. Pengujian *variable* pada data *test sthamer triangle* juga sama, menunjukkan hasil yang sesuai dengan jalannya metode *Particle Swarm Optimation*, terbukti juga data *test sthamer triangle* juga menghasilkan data *test* atau *variable* dengan angka yang paling kecil dan *kualifikasi* hasil segitiga.

ABSTRACT

Samudra, Alifiyah Pusaka Dewi. 2020. ***Particle Swarm Optimization Algorithm 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) Ainatul Mardhiyah, M.Cs.

Key Word : *Particle Swarm Optimization, Test Data, Software*

Technology in the field of quality software has been widely created. However, the development of software quality sometimes does not go through the most important process of software testing. System testing procedures cannot be done carelessly, but rather have a test scenario or framework created by system testers. Software testing can be done by generating test data, namely using myers triangle and sthamer triangle test data as test or input data, the first step is to convert the test data into Control Flow Graph (CFG) which then becomes the path or node path after it is processed using the Particle Swarm Optimation method to find the final result or output value of variables A, B, and C that become the side values of the triangle. The test results of myers triangle variables get the smallest variable value compared to other variables, and it is proven that the Particle Swarm Optimation method works very precisely and produces qualification of the scalene triangle results. Variable testing in sthamer triangle test data is also the same, showing results that are in accordance with the course of Particle Swarm Optimation method, it is also proven that sthamer triangle test data also produces test data or variables with the least number and qualification of triangle results.

الملخص

سامودرا، ألفية فوساكا ديوي. 2020. الخوارزمية الذرة السربية الأقوية لإستفزاز بيانة الإختبار أوتوماتيكيا في إختبار البرمجيات. البحث العلمي. قسم المعلوماتية كلية العلوم والتكنولوجية جامعة مولانا مالك إبراهيم الإسلامية الحكومية مالانج. المشرف: 1) الحاج فتح الرحمن الماجستير. 2) أعينة المرضية الماجستير.

الكلمات المفاتح : الذرة السربية الأقوية، يانة الإختبار، البرمجيات.

مستخلص البحث: خلقت التكنولوجية في البرمجيات المزية كثيرا. بل، لا تمر تنمية مزية البرمجيات العملية الأهمية أحيانا هي إختبار البرمجيات. لايستطيع منهاج إختبار النظام ان يفعل عشوائيا. بل، يملك السيناريو أو إطار الإختبار الذي يجعل مختبر النظام. يستطيع إختبار البرمجيات ان يفعل بإستفزاز بيانات الإختبار، هو يستخدم بيانات الإختبار المثلث مايرز (test myers triangle) والمثلث الستامر (sthamer triangle) بيانات الإختبار أوالإدخال (inputan)، مرحلة الأولى بتغير بيانات الإختبار إلى الرسم البياني للتحكم في التدفق (Control Flow Graph) الذي يصبح سبيل المسار (path) أو الأقدة (node) بعد ذالك. ثم، يفعل العملية باستخدام طريقة تحسين سرب الجسيمات (Particle Swarm Optimation) لطلب الحصيلة الأخيرة أو إخراج القيمة من التغير أ، ب، و ج التي تصبح الأنحاء من المثلث. نالت حصيلة إختبار التغير المثلث مايرز (test myers triangle) القيمة التغيرة أصغر من الأخرى، وتتأكد بطريقة تحسين سرب الجسيمات (Particle Swarm Optimation) تعمل صحيحا شديدا وتحصل إستحقاق حصيلة المثلث على نقيض الطويل. إختبار المتغير في بيانات الإختبار المثلث مايرز (test myers triangle) متساويا أيضا، يدل الحصيلة المناسبة بأداء طريقة تحسين سرب الجسيمات (Particle Swarm Optimation) ، تأكد أيضا أن بيانات الإختبار المثلث مايرز (test myers triangle) تحصل بيانات الإختبار أو التغير بالعدد الأصغروالإستحقاقحصيلةالمثلث.

BAB I

PENDAHULUAN

Pada bab ini akan dijelaskan 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. Tujuan penelitian berisi tujuan dilakukannya penelitian ini dan batasan penelitian berisi batasan pada penelitian agar penelitian tidak meluas dari focus dilakukannya penelitian ini.

1.1 Latar Belakang Masalah

Pada zaman sekarang teknologi sudah semakin maju, salah satunya dibidang pembuatan perangkat lunak. Banyak perangkat lunak berkualitas yang sudah diciptakan. 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 sistem merupakan tahap akhir dalam rangkaian pengembangan perangkat lunak. Pada tahap tersebut, 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. Pengujian secara konvensional atau menggunakan jasa penguji sistem membutuhkan biaya produksi tetap, karena tahap

pengujian harus dilakukan kembali setiap ada pengembangan *(update)* perangkat lunak.

Salah satu pendekatan atau kerangka pengujian sistem adalah *white box testing. White box testing* merupakan pengujian struktural atau *code* sistem. Pada *white box testing* terdapat beberapa ruang lingkup yaitu *statement coverage, branch coverage, basis path coverage* dan *fault based coverage.* Diantara beberapa ruang lingkup pengujian *white box, branch coverage* merupakan pengujian yang peling efektif dalam menekan biaya pengujian (Mao, 2014). *Branch Coverage* merupakan pengujian pada area *code* percabangan. Pada umumnya pengujian pada ruang lingkup *branch coverage*, penguji menginputkan data pada *code* percabangan untuk mengetahui respon yang didapat dari sistem. Apabila respon *code* sesuai dengan target pengujian, maka *code* berjalan dengan benar.

Salah satu upaya untuk meneka biaya produksi tetap, maka dilakukan pengujian secara otomatis dengan menggunakan algoritma cerdas (*Artificial Intelligence)* seperti genetika, *Simulated Annealing (SA), dan Particle Swarm Optimization.* Masing-masing metode memiliki kelebihan dan kelemahan, Algorima *SA* pada umumnya berjalan optimal untuk permasalahan yang kompleks namun membutuhkan waktu proses yang lebih lama dibanding algoritma lainnya. Begitu juga algoritma genetika dapat berjalan untuk persoalan data dalam jumlahbesar namun memiliki alur proses yang rumit sehingga membutuhkan waktu proses yang lebih lama.

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

الَّذِي خَلَقَ الْمَوْتَ وَالْحَيَاةَ لِيَبْلُوَكُمْ أَيُّكُمْ أَحْسَنُ عَمَلا وَهُوَ الْعَزِيزُ الْغَفُورُ

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

Tafsir Ibnu Katsir menjelaskan tentang *QS Al-Mulk* pada potongan ayat.

(الَّذِي خَلَقَ الْمَوْتَ وَالْحَيَاةَ لِيَبْلُوَكُمْ)

"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 bahwaAllah 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,

كَيْفَ تَكْفُرُونَ بِاللَّهِ وَكُنْتُمْ أَمْوَاتًا فَأَحْيَاكُمْ

"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(kema tian). Sedangkan penciptaan disebut hayat (kehidupan). Oleh karena itu, Allah Ta'ala (لِيَبْلُوَكُمْ أَيُّكُمْ أَحْسَنُ عَمَلًا ۚ) "Supaya Dia mengujimu, supaya diantara kamu yang lebik baik amalnya." Yakni, yang paling baik amalnya, sebagaimana yangdikatakan oleh Muhammad bin Ajlan. Dan Alla 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 data *test* yang baik.

Berdasarkan penelitian yang dilakukan oleh (Shi, Y, 1998) Algoritma *Particle Swarm Optimization* terbukti optimal untuk membangkitkan data uji pada permasalahan pengujian sistem. Begitu juga penelitian yang dilakukan oleh (Kennedy, 1995) membuktikan Algoritma *Particle Swarm Optimization* berjalan dengan baik pada proses pengujian sistem secara otomatis. Kedua penelitian merekomendasikan algoritma *Particle Swarm Optimization* karena memiliki proses komputasi yang lebih sedikit. Apabila dibandingkan dengan algortima genetika, algortima *Particle Swarm Optimization* tidak membutuhkan proses *mutasi* dan *crossover*. Hal ini membuat algoritma *Particle Swarm Optimization* memiliki waktu proses komputasi yang lebih baik. Algoritma *Particle Swarm Optimization* memiliki dasar pemikiran yang memperhatikan perilaku kelompok atau kawanan hewan seperti burung, semut dan lebah. Masing-masing kelompok hewan pada algoritma ini disebut *particle*. Setiap *particle* memiliki sifat atau perilaku yang berbeda-beda, sehingga Algoritma *Particle Swarm Optimization* menggabungkan setiap particle untuk menyelesaikan masalah. Pada penelitian ini Algoritma *Particle Swarm Optimization* akan membangkitkan data *test* yang akan menghasilkan nilai *variable* secara ototmatis agar memakan waktu yang lebih singkat dan tidak ada kesalahan dalam pembangkitan nilai *variable*.

1.2 Identifikasi Masalah

Berdasarkan uraian latar belakang penelitian yang telah diuraikan, maka rumusan masalah pada penelitian ini adalah Bagaimana cara membangkitkan data *test* secara otomatis dengan menggunakan algoritma *Particle Swarm Optimization*?

1.3 Tujuan Penelitian

Penelitian ini dilakukan dengan tujuan untuk membangkitkan data *test* pengujian dengan menggunakan algoritma *Particle Swarm Optimization*.

1.4 Manfaat Penelitian

Sedangkan manfaat penelitian ini adalah Memberikan kemudahan bagi penguji dalam pengujian data tes secara otomatis, sehingga dapat mempersingkat waktu untuk melakukan pengujian yang lainnya.

1.5 Batasan Masalah

Penelitian ini menerapkan batasan masalah, agar hasil penelitian berfokus pada permasalahan yang berhasil di identifikasi. Batasan masalah pada penelitian ini adalah :

Menggunakan *branchmark* data *test myers triangle* dan *sthamer triangle*

1. *Source code* pemrograman berbasis java.
2. Pengujian data *test* dilakukan dengan menggunakan *software netbeans.*

1.6 Sistematika Penulisan

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

BAB I : PENDAHULUAN

Bab ini berisi tentang latar belakang dilakukannya penelitian, rumusan masalah yang dibahas dalam penelitian, batasan masalah yang dibutuhkan

pada penelitian, tujuan dilakukan penelitian, manfaat yang diterima dari penelitian, sistematika penulisan pada laporan penelitian.

BAB II : TINJAUAN PUSTAKA

Bab ini menjelaskan bagaimana penelitian yang telah dilakukan diantaranya teori dasar dan data-data yang terkait dengan pembangkitan data *test*.

BAB III : METODOLOGI PENELITIAN

Bab ini berisi tentang prosedur atau pembangkitan data *test* secara otomatis serta implementasi *particle swarm optimization* pada pengujian perangkat lunak.

BAB IV : HASIL DAN PEMBAHASAN

Bab ini menjelaskan pengujian dari pembangkitan data *test* yang dibangkitkan, yang selanjutnya dilakukan pembahasan secara terperinci dan proses pengujian tersebut. Hasil dan pengujian akan dijelaskan secara terperinci dengan menyertakan gambar setiap proses pengujian.

BAB V : PENUTUP

Pada bab ini berisi kesimpulan dari penelitian yang telah dilakukan dan saran yang berguna untuk penelitian selanjutnya.

BAB II

TINJAUAN PUSTAKA

2.1 Data

Data adalah fakta yang sudah ditulis dalam bentuk catatan atau direkam ke dalam berbagai bentuk media, seperti buku catatan atau komputer. Data merupakan fakta yang belum memiliki makna (McMinn P, 2004). Bentuk ini belum dapat memberikan manfaat besar bagi penerimanya, sehingga memerlukan model yang nantinya akan dikelompokkan dan di proses untuk menghasilkan informasi. Data dapat diklasifikasikan berdasarkann jenis, sifat, sumber, dan isinya.

1. Berdasarkan jenisnya, data dibagi menjadi data statis (data yang tidak atau jarang berubah) dan data dinamis (data yang selalu berubah dan memerlukan update per periodenya).
2. Berdasarkan sifatnya, dikenal data kuantitatif (daya dengan hitungan bilangan) dan kualitatif (data yang tidak di hitung dengan bilangn, tapi diukur dengan kata-kata yang bernilai.
3. Berdasarkan sumbernya, data dikelompokkan dalam data internal (data yang didapatkan dari dalam perusahaan atau organisasi yang bersangkutan) dan data eksternal (data yang berasal dari luar perusahaan).
4. Berdasarkan isinya, data dibagi menjadi data catatan kegiatan, data hasil penelitian, data lingkungan dan data peraturan.

   Jika dilihat dari jenisnya, maka data dibedakan menjadi :

1) Data primer, yaitu berupa teks hasil wawancara dan diperoleh melalui wawancara dengan informan yang sedang dijadikan sampel dalam penelitiannya. Data dapat direkam atau dicatat oleh peneliti.
2) Data sekunder, yaitu data yang berupa data-data yang sudah tersedia dan dapat diperoleh oleh peneliti dengan membaca, melihat atau mendengarkan. Data ini biasnya berasal dari data primer yang sudah diolah oleh peneliti sebelumnya. Contohnya : dokumen, surat, foto, rekaman kaset, video, dan sebagainya.

2.2 Data *Test*

Data *test* adalah data yang digunakan dalam pengujian sistem perangkat lunak. Untuk menguji aplikasi perangkat lunak, perlu dimasukkan beberapa data untuk menguji sebagian besar fitur. Setiap data khusus yang diidentifikasi yang digunakan dalam pengujian dikenal sebagai data *test*. Data *test* dapat berupa lembar excel yang dapat dimasukkan secara manual saat menjalankan uji kasus atau dapat dibaca secara otomatis dari file (XML, *flat file*, basis data, dan lain-lain) menggunakan *tool*.

Data *test* digunakan untuk melakukan konfirmasi terhadap hasil yang diharapkan. Misalnya ketika sejumlah data *test* dimasukkan, apakah hasil yang diharapkan sesuai dengan yang ditampilkan dan apa yang terjadi jika program mendapat input yang salah. Data *test* dapat dihasilkan secara manual oleh penguji atau menggunakan *tool* otomatis yang mendukung pengujian (Offut, A Jefferson, 1999).

2.3 Pengertian 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 (Hla, K.H.S, 2008). Sebuah perangkat lunak harus dapat diprediksi dan konsisten, hal tersebut dilakukan untuk menghasilkan perangkat lunak yang baik.

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 (Pressman, 2010). 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 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 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 share dapat dipahami dengan baik. Dokumentasi teknis dapat langsung diakses, terorganisasi dengan baik, spesifik dan rinci, serta akurat. Perubahan desain harus dikomunikasikan kepada penguji.

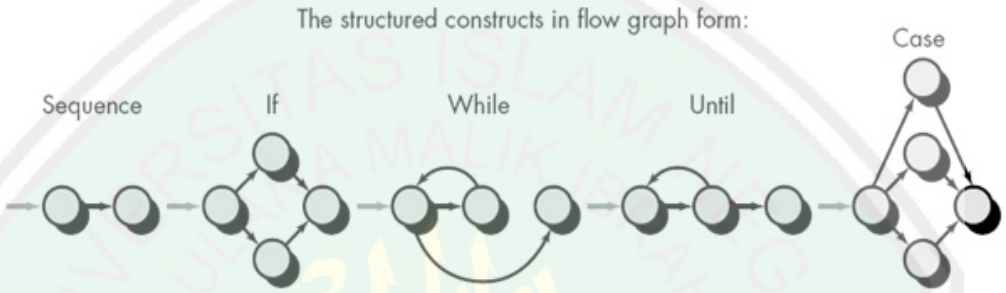
2.4 Path Testing

Path Testing adalah metode pengujian yang memungkinkan desainer *test case* mengukur kompleksitas logika desain posedural dan menggunakannya sebagai pedoman untuk menetapkan basis set dari jalur eksekusi. Metode ini menjamin bahwa setiap statement akan dilalui minimal sekali dalam proses pengujian. Tujuannya adalah meyakinkan bahwa himpunan *test case* akan menguji setiap *path* pada suatu program paling sedikit satu kali. Titik awal *path testing* adalah suatu program *flow graph* yang menunjukkan *node-node* yang menyatakan keputusan program, misalnya: kondisi *if-then-else* dan busur yang menyatakan alur kontrol. Statement yang memiliki kondisi ada pada *node-node* dalam *flow graph*. *Path testing* menggambarkan alur kontrol di mana setiap cabang ditunjukkan oleh *path* yang terpisah sedangkan *loop* ditunjukkan oleh *arrows looping* dan kembali ke *loop* kondisi *node* (Sommerville, 2003).

2.5 Control Flow Graph

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 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:



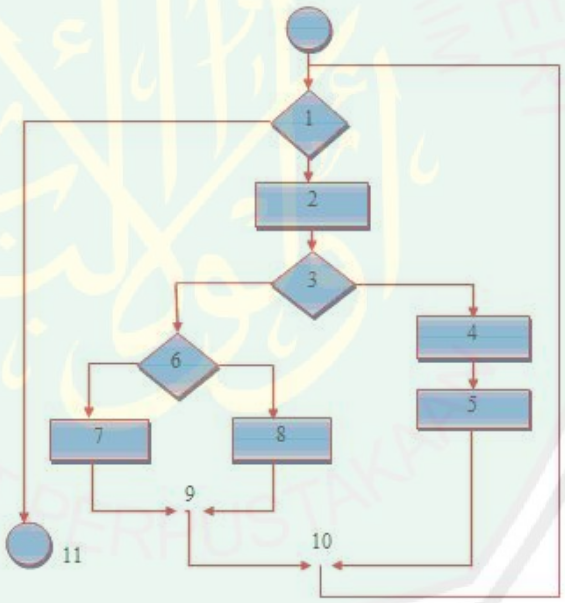
Gambar 2. 1 Notasi struktur kontrol pada *flow graph*

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

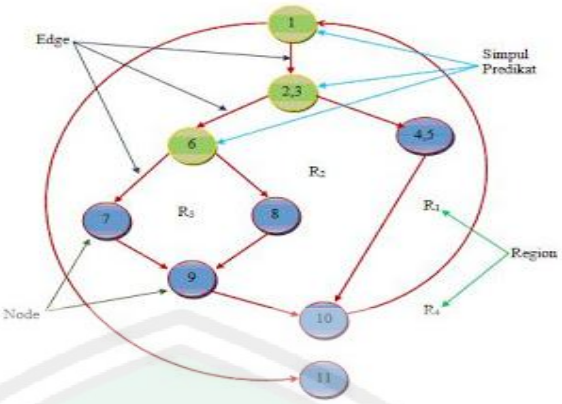
1. Satu atau lebih statement secara sekuensial yang dikelompokkan
2. Percabangan seleksi dari satu statement kedua pilihan statement (seleksi)
3. Penggabungan dua statement yang dilanjutkan pada satu statement yang sama (*merge*)

Sedangkan notasi garis panah disebut sebagai *edge* atau *link*, menggambarkan aliran kontrol. Setiap *edge* harus dihubungkan dari sebuah *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 desain *language source code* yang telah dibuat sebelumnya.

Untuk menggambarkan *flowgraph* dapat menggunakan simpul *flowgraph* yang berbentuk lingkaran untuk mempresentasikan satu atau lebih statemen prosedural. Anak panah pada *flowgraph* disebut *edges* atau *links*, untuk mempresentasikan aliran kontrol dan analog dengan panah bagan alur. *Edge* harus berhenti pada suatu simpul, meskipun bila simpul tersebut tidak mempresentasikan statemen prosedural (misalnya, lihat simpul untuk bangun *ifthen-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 2.3.



Gambar 2. 2 Contoh *flowchart* (Sumber : Roger S. Pressman, 2002)



Gambar 2. 3 Transformasi *flowchart* ke *flowgraph* (Sumber : Roger S. Pressman, 2002)

2.6 *Cyclomatic Complexity*

Cyclomatic Complexity adalah sebuah *software metric* yang menyediakan ukuran kuantitatif dari kompleksitas logika dari sebuah program. Dengan menggunakan hasil pengukuran atau perhitungan dari *metric cyclomatic complexity*, kita dapat menentukan apakah sebuah program merupakan program yang sederhana atau kompleks berdasarkan logika yang diterapkan pada program tersebut. Apabila dikaitkan dengan pengujian perangkat lunak (*software testing*), *cyclomatic complexity* dapat digunakan untuk menentukan berapa minimal *test case* yang harus dijalankan untuk menguji sebuah program dengan menggunakan teknik *basis path testing*. Pada pengujian *basis path*, aliran control logika digambarkan dengan menggunakan *flow graph* (Pressman, 2002)

Dari *flow graph* yang sudah tersedia pada Gambar 2.3, *cyclomatic complexity* dari sebuah program dapat dibuat dengan menggunakan rumus dibawah ini:

$$V(G) = E - N + 2 \qquad (2.1)$$

V(G) : *cyclomatic complexity*

E : total jumlah *edge*

N : total jumlah *node*

Hasil *independent path* pada Gambar 2.3 dapat dijabarkan sebagai berikut:

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

Pada contoh *flow graph* di Gambar 2.3, dapat dihitung *cyclomatic complexity*-nya sebagai berikut:

V(G) = 11 *edges* – 9 *nodes* + 2 = 4

Angka 4 dari hasil perhitungan *cyclomatic complexity* menunjukan jumlah independent *path* dari basis *path testing*, atau dengan kata lain menunjukkan jumlah pengujian yang harus dijalankan untuk memastikan semua statement pada program dijalankan minimal sekali (semua statement telah diuji).

Catatan:

1. *independent path* adalah setiap *path* yang dilalui program yang menunjukkan satu set baru dari pemrosesan statement atau dari sebuah kondisi baru
2. *independent path* pada *flow graph* harus melewati sedikitnya satu *edge* yang belum pernah dilewati oleh *path* sebelumnya
3. *independent path* selalu dimulai dari *node* awal hingga ke *node* terakhir
4. *independent path* yang dibuat pertama kali adalah *independent path* terpendek

2.7 *Particle Swarm Optimization*

Particle Swarm Optimization diperkenalkan oleh Dr. Eberhart dan Dr. Kennedy pada tahun 1995, merupakan algoritma optimasi yang meniru proses yang terjadi dalam kehidupan populasi burung (*lock of bird*) dan ikan (*school of fish*) dalam bertahan hidup. Sejak diperkenalkan pertama kali, algoritma *Particle Swarm Optimization* berkembang cukup pesat, baik dari sisi aplikasi maupun dari sisi pengembangan metode yang digunakan pada algoritma *Particle Swarm Optimization* tersebut. Oleh sebab itu hal tersebut, mereka mengkategorikan algoritma sebagai bagian dari kehidupan rekayasa/buatan *Arificial Life*. Algorima ini juga terhubung dengan komputasi *evolusioner*, algoritma *genetic* dan pemrograman *evolusionari* (Poli, R, 2008).

Particle Swarm Optimization, sekumpulan dimaksud memiliki dimensi tersendiri dengan setiap partikel letak awal mulanya terdapat disebuah tempat yang random dalam ruang multidimensi. Setiap partikel diibaratkan memiliki dua ciri yaitu letak serta kecepatan. tiap partikel berpindah dalam ruang ataupun *space* tertentu serta mengingat letak terbaik yang sempat dilewati ataupun ditemui terhadap asal makanan ataupun nilai fungsi obyektif. Tiap partikel membagikan informasi atau letak terbaiknya kepada partikel yang lain untuk mebiasakan letak serta kecepatan masing-masing berdasarkan informasi yang diterima mengenai letak yang bagus tersebut. *Particle Swarm Optimization* merupakan sebagian dari metode komputasi *evolusioner*, yang mana populasi pada *Particle Swarm Optimization* pada dasar penelusuran algoritma serta dimulai oleh sesuatu populasi yang random yang disebut dengan partikel. Berbeda dengan metode komputasi evolusioner yang lain, tiap partikel di *Particle Swarm Optimization*

selalu berhubungan dengan sesuatu *velocity*. Partikel-partikel tersebut berpindah melalui pencarian ruang dengan *velocity* yang dinamis yang dipaskan menurut individu historianya, karena itu, partikel-partikel mengarah untuk berpindah ke zona pencarian yang lebih bagus sesudah melalui proses pencarian. *Particle Swarm Optimization* memiliki keselarasan serupa *genetic algorithm* yang mana diawali dengan suatu populasi yang acak dalam wujud *matriks*. Tetapi *Particle Swarm Optimization* tidak ada operator evolusi yakni *crossover* serta *mutase* semacam yang terdapat di *genetic algorithm*. Baris pada *matriks* disebut partikel ataupun dalam *genetic algorithm* disebut *kromosom* yang terdiri dari nilai suatu *variable*. Tiap partikel bergerak dari letak awal ke posisi yang baik dengan suatu *velocity*.

Pada algoritma *Particle Swarm Optimization* vektor *velocity* di perbarui untuk tiap-tiap partikel setelah itu menambahkan vektor *velocity* tersebut ke letak *particle*. *Update velocity* dipengaruhi oleh kedua penyelesaian yaitu *global best* yang berhubungan dengan pengeluaran yang sangat kecil yang sempat didapat di suatu partikel serta solusi *local best* yang berkaitan dengan anggaran yang sangat rendah pada populasi awal. Jika solusi *local best* memiliki suatu anggaran yang kurang dari anggaran solusi *global* yang ada, maka solusi *local best* menggantikan solusi *global best*. Kemudahan algoritma serta performan yang baik, membuat *Particle Swarm Optimization* sudah memikat banyak atensi di golongan periset serta sudah di aplikasikan dengan berbagai masalah optimasi. *Particle Swarm Optimization* sudah terkenal menjadi optimisasi *global* dengan sebagian besar permasalahan yang bisa dituntaskan dengan bagus dimana variable-variabelnya merupakan bilangan riil. Menurut (Chen, X, 2010), sebagian kata umum yang

sering digunakan dalam *Particle Swarm Optimization* bisa diartikan sebagai berikut:

a. *Swarm* : Populasi dari algoritma
b. *Particle* : Anggota (individu) pada *swarm*. Tiap *particle* menjelaskan suatu solusi yang potensial di kasus yang diselesaikan. Letak dari suatu *particle* ialah ditentukan oleh representasi solusi saat itu. *Pbest (Persolan Best)* : posisi *Pbest* sebuah *particle* yang menampilkan letak *particle* yang dipersiapkan untuk memperoleh suatu solusi yang terbaik.
c. *Gbest* (*Global best*) : posisi terbaik *particle* pada *swarm* ataupun posisi terbaik diantara *Pbest* yang ada.
d. *Velocity* (v): vektor yang menggerakkan proses optimisasi yang memutuskan arah di mana suatu *particle* dibutuhkan untuk berpindah (*move*) untuk memperbaiki posisinya semula ataupun kecepatan yang menggerakkan proses optimasi yang memastikan arah dimana *particle* dibutuhkan untuk berpindah serta memperbaiki posisinya semula.
e. *Inertia weight* (θ): *inertia weight* di simbolkan w, parameter ini digunakan untuk mengendalikan akibat dari adanya *velocity* yang diberikan oleh suatu *particle*.
f. *Learning Rates* (c1 dan c2) : suatu konstanta untuk menilai keahlian *particle* (c1) serta keahlian sosial *swarm* (c2) yang menampilkan bobot dari *particle* terhadap memorinya.

Menurut Chen & Shih (2013) letak pada masing-masing partikel dapat disebut sebagai calon solusi (*candidate solution*) untuk suatu masalah optimisasi. Masing-masing partikel diberi suatu fungsi *fitness* merancang sesuai dengan

menunjuk permasalahan yang bersepadan. Ketika tiap-tiap partikel bergerak ke suatu posisi baru didalam ruang pencarian, itu akan mengingat sebagai personal *best* (*Pbest*). Menjadi tambahan terhadap ingatan informasi sendiri, tiap-tiap partikel juga menukar informasi dengan partikel yang lain serta mengingat *global best* (*Gbest*). Setelah itu tiap-tiap partikel akan melihat kembali arah serta kecepatannya sesuai dengan *Pbest* serta *Gbest* untuk bergeser ke arah yang optimal dan menemukan solusi yang optimal. Dengan keuntungan dari aplikasi yang mudah serta sederhana, sedikit parameter yang dibutuhkan, dengan hasil yang baik, *Particle Swarm Optimization* telah diangkat banyak dalam bidang, semacam *TSP, flowshop, VRP, task-resource assignment*, penjadwalan khusus dan lain lain. Karena itu *Particle Swarm Optimization* sudah digunakan dalam membentuk penjadwalan yang optimal untuk *university cources*. Seperti contoh dengan algoritma evolusioner yang lain, algoritma *Particle Swarm Optimization* merupakan suatu populasi yang didasarkan penelusuran inisialisasi partikel secara random serta terdapat interaksi diantara partikel dalam populasi. Di dalam *Particle Swarm Optimization* setiap partikel bergerak melalui ruang solusi dan mempunyai kemampuan untuk mengingat posisi terbaik sebelumnya serta bisa bertahan dari generasi ke generasi. Algoritma *Particle Swarm Optimization* dikembangkan dengan berdasarkan pada model berikut:

a. Pada saat seekor burung mendekati sasaran atau makanan (atau bisa minimum atau maximum suatu fungsi tujuan) secara cepat mengirim informasi kepada burung-burung yang lain dalam sekumpulan tertentu.
b. Burung yang lain akan mengikuti arah menuju ke makanan namun tidak secara langsung.

c. Terdapat komponen yang bergantung pada pikiran setiap burung, ialah memorinya tentang apa yang telah dilewati pada waktu sebelumnya.

Keuntunggan dari Algoritma *Particle Swarm Optimization* adalah:

a. *Particle Swarm Optimization* berdasar pada kecerdasan (*intelligence*). Ini bisa diterapkan ke dalam kedua penggunaan dalam bidang metode dan riset ilmiah.
b. *Particle Swarm Optimization* tidak memiliki *overlap* serta kalkulasi mutasi. Pencarian dapat dilakukan oleh kecepatan dari partikel. Selama pengembangan beberapa generasi, mayoritas hanya partikel yang optimis yang bisa mengirim informasi kepartikel yang lain, serta kecepatan dari pencarian adalah sangat cepat.
c. Perhitungan didalam Algoritma *Particle Swarm Optimization* sangat mudah, menggunakan kemampuan optimisasi yang lebih besar serta bisa diselesaikan dengan mudah.
d. *Particle Swarm Optimization* mengunakan kode/jumlah riil, dengan diputuskan langsung dengan solusi, serta total dimensi sama dengan solusi yang ada.

Beberapa kerugian dari Algoritma *Particle Swarm Optimization* adalah:

a. Metode mudah mendapatkan optimal parsial (sebagian), yang mana menyebabkan semakin sedikit ketetapannya untuk peraturan tentang arah dan kecepatan.
b. Metode tidak bisa berkembang dari permasalahan sistem yang tidak terkonfirmasi, seperti solusi dalam bidang energi dan peraturan yang tidak menentu dialami bidang energi.

Model Algoritma Particle Swarm Optimization ini hendak disimulasikan dalam ruang dengan ukuran tertentu dengan beberapa iterasi sehingga di tiap iterasi, posisi partikel hendak terus menjadi menuju ke sasaran yang dituju(minimasi ataupun maksimasi guna). Ini dicoba sampai maksimum iterasi dicapai ataupun dapat pula digunakan kriteria penghentian yang lain. Perihal ini diakibatkan, Particle Swarm Optimization ialah algoritma optimasi yang gampang dimengerti, lumayan simpel, serta mempunyai unjuk kerja yang telah teruji profesional. Algoritma Particle Swarm Optimization bisa digunakan pada bermacam permasalahan optimasi baik kontinyu ataupun diskrit, linier ataupun nonlinier. Particle Swarm Optimization mencerminkan kegiatan penelitian pemecahan terbaik di sesuatu tempat pemecahan bagaikan kegiatan terbangnya kelompok partikel di sesuatu tempat pemecahan tersebut. Dengan demikian, dini pencarian pada algoritma Particle Swarm Optimization dicoba dengan populasi yang random(acak) yang diucap dengan partikel serta bila sesuatu partikel ataupun seekor burung menciptakan jalur yang pas ataupun pendek mengarah sumber santapan, hingga sisa kelompok yang lain pula hendak lekas menjajaki jalur tersebut walaupun posisi mereka jauh di kelompok tersebut. Posisi partikel dalam ruang pemecahan tersebut ialah kandidat pemecahan yang berisi variabel-variabel optimasi. Tiap posisi tersebut hendak berhubungan dengan suatu nilai yang diucap nilai objektif ataupun nilai fitness yang dihitung bersumber pada guna objektif dari permasalahan optimasi yang hendak dituntaskan.

2.7.1 Proses *Particle Swarm Optimization*

Menurut Chen & Shih (2013) untuk memulai algoritma *Particle Swarm Optimization*, kecepatan awal (*velocity*) dan posisi awal (*position*) ditentukan secara acak. Kemudian proses pengembangan sebagai berikut :

a. Asumsikan kalua dimensi kelompok ataupun sekumpulan (jumlah partikel) merupakan N. Kecepatan serta posisi dini pada masing-masing partikel dalam N ukuran ditetapkan secara random (acak).
b. Hitung kecepatan dari seluruh partikel. Seluruh partikel bergerak mengarah titik optimal dengan suatu kecepatan. Awalnya semua kecepatan dari partikel diasumsikan sama dengan nol, set iterasi i = 1.
c. Nilai *fitness* setiap partikel ditaksir menurut fungsi sasaran (*objective function*) yang ditetapkan. Jika nilai *fitness* setiap partikel pada lokasi saat ini lebih baik dari *Pbest*, maka *Pbest* diatur untuk posisi saat ini.
d. Nilai *fitness* partikel dibandingkan dengan *Gbest*. Jika *Gbest* yang terbaik maka *Gbest* yang diupdate.
e. Persamaan (2.1) dan (2.2) ditunjukkan di bawah ini untuk memperbaharui (*update*) kecepatan (*velocity*) dan posisi (*position*) setiap partikel.

$$V_{id}^{k+1} = w \times V_{id}^{k} + c1 \times rand1 \times (P_{id} - X_{id}) + c2 \times rand2 \times (G_{id} - X_{id}) \quad (2.2)$$

$$X_{id}^{k+1} = X_{id}^{k} + V_{id}^{k+1} \quad (2.3)$$

Dimana :

Vid = komponen kecepatan individu ke i pada d dimensi

Xid = posisi individu i pada d dimensi

ω = parameter inertia *weight*

$c1$ $c2$ = konstanta akselerasi (*learning rate*), nilainya antara 0 sampai 1

rand1, rand2 = parameter *random* antara 0 sampai 1

*Pi*d= *Pbest* (*local best*) individu i pada d dimensi

*G*id= *Gbest* (*global best*) pada d dimensi

Langkah-langkahnya:

1) Cek apakah solusi yang sekarang sudah konvergen. Jika posisi semua partikel menuju ke satu nilai yang sama, maka ini disebut konvergen. Jika belum konvergen maka langkah 2 diulang dengan memperbarui iterasi i = i + 1, dengan cara menghitung nilai baru dari *Pbest* dan *Gbest*. Proses iterasi ini dilanjutkan sampai semua partikel menuju ke satu titik solusi yang sama. Biasanya akan ditentukan dengan kriteria penghentian (*stopping criteria*), misalnya jumlah selisih solusi sekarang dengan solusi sebelumnya sudah sangat kecil.
2) Ada dua aspek penting dalam memilih kondisi berhenti yaitu :
   a) Kondisi berhenti tapi tidak menimbulkan *Particle Swarm Optimization convergent premature* (memusat sebelum waktunya) dimana solusi tidak optimal yang didapat.
   b) Kondisi berhenti harus melindungi dari kondisi oversampling pada nilainya, jika kondisi berhenti memerlukan perhitungan yang terus menerus maka kerumitan dari proses pencarian akan meningkat.

Beberapa kondisi berhenti yang dapat dipakai dalam *Particle Swarm Optimization* adalah berhenti ketika jumlah iterasi telah mencapai jumlah maksimum yang diperoleh, berhenti ketika solusi yang didapatkan ditemukan, berhenti ketika tidak ada perkembangan setelah beberapa iterasi.

2.8 Penelitian Terkait

Penelitian terkait ini menjadi salah satu acuan peneliti dalam melakukan penelitian sehingga peneliti dapat memperbanyak teori yang digunakan dalam mengkaji penelitian yang dilakukan. Dari penelitian terkait peneliti mengangkat beberapa penelitian sebagai refrensi dalam memperbanyak bahan kajian pada penelitian peneliti. Berikut merupakan peneltian terkait beberapa jurnal terkait dengan penelitian yang dilakukan peneliti.

Alshraideh, dkk (2011) membahas tentang pengusulan proses pembangkitan data *test* menggunakan *multiple-population* pada algoritma genetika. 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 *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*.

Maulana Reza, dkk (2015) membahas tentang 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 genetika (AG). Salah satu permasalahan AG dalam menghasilkan data *test* 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. Oleh karena itu di dalam penelitian ini diusulkan

integrasi pareto *fitness*, *multiple-population* dan *temporary population* di dalam proses pencarian data *test* dengan menggunakan AG (AG-PFMPTP). *Multiple-population* diusulkan untuk menghindari *premature convergence*. Kemudian pareto *fitness* dan *temporary population* digunakan untuk mencari beberapa data *test* sekaligus, kemudian mengevaluasinya dan memasukkan ke dalam *archive temporary population*. Dari hasil pengujian yang telah dilakukan rata-rata generasi metode AG-PFMPTP secara signifikan lebih sedikit dalam menghasilkan data *test* yang dibutuhkan dibandingkan metode AG standar ataupun AG dengan *multiple-population* (AG-MP) pada semua *benchmark* program yang digunakan. Hal tersebut menunjukkan metode yang diusulkan lebih cepat dalam mencari data *test* yang dibutuhkan.

Fabio Palomba (2016) membahas tentang Pengujian perangkat lunak yang kompleks secara keseluruhan akan memakan waktu yang lama dan membutuhkan sumber daya manusia yang banyak. Mengotomasi bagian dari pengujian akan membuat proses ini menjadi lebih cepat dan mengurangi kerawanan akan kesalahan. Pada penelitian ini, telah dibangun 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.

Mandyartha (2017) membahas tentang 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.

BAB III

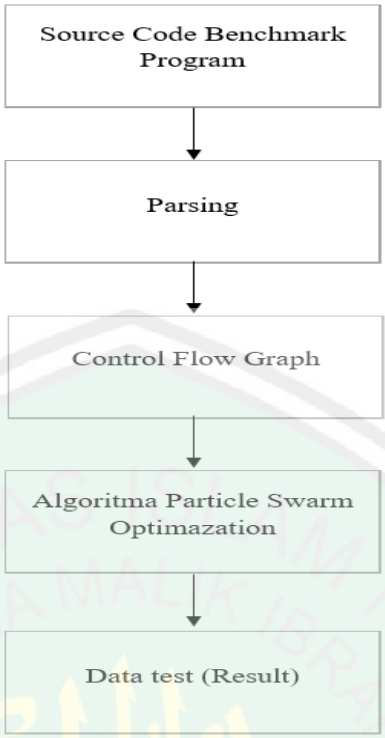
METODOLOGI PENELITIAN

Pada bab ini membahas mengenai beberapa hal, yaitu tahap penelitian yang akan dilakukan, kebutuhan system yang akan dibuat, serta penyelesaian masalah. Pembangkitan data *test* pada pengujian perangkat lunak menggunakan metode algoritma *particle swarm optimization*.

3.1 Perancangan Sistem

Perancangan sistem adalah merancang atau mendesain suatu system yang baik yang isinya adalah langkah-langkah operasi dalam proses pengolahan data dan proses prosedur-prosedur untuk mendukung operasi sistem. Tujuan dari perancangan sistem adalah untuk memenuhi kebutuhan para pemakai sistem serta memberikan gambaran yang jelas dan rancang bangun yang lengkap kepada programmer dan ahli-ahli yang terlibat didalam.

Pada bagian ini menjelaskan langkah-langkah system yang akan dibangun. Adapun langkah-langkah penyusunan system yang akan dibangun meliputi : masukan (*input*) berupa fungsi dari program kemudian di analasis menggunakan *control flow graph* (*CFG*) yang menunjukan aliran (*path*) kode program. Setelah aliran (*path*) kode program diketahui kemudian dibangkitkan data uji dengan metode algoritma *particle swarm optimization*. Langkah desain system ini dapat dilihat pada Gambar 3.1.



Gambar 3. 1 Desain Sistem

3.2 *Source Code Benchmark* Program

Dalam komputasi, kode sumber adalah kumpulan kode apa saja, mungkin dengan komentar, ditulis menggunakan bahasa pemrograman yang dapat dibaca manusia, biasanya sebagai teks biasa. Kode sumber suatu program dirancang khusus untuk memudahkan pekerjaan pemrogram komputer, yang menentukan tindakan yang akan dilakukan oleh komputer kebanyakan dengan menulis kode sumber. Kode sumber sering diubah oleh *assembler* atau *kompiler* menjadi kode mesin biner yang dipahami oleh komputer. Kode mesin kemudian dapat disimpan untuk dieksekusi di lain waktu. Atau, kode sumber dapat ditafsirkan dan dengan demikian segera dieksekusi (T Manikumar dkk, 2016).

Pada penelitian ini menggunakan *source code benchmark* data program yang sudah banyak digunakan oleh peneliti lain seperti pachauri et al. dan ferrer et

al. Pada penelitian ini hanya menggunakan satu *benchmark* program yaitu *Myers Triangel*. *Branchmark* program tersebut didapatkan dari situs : http://shodhganga.inflibnet.ac.in/bitstream/10603/22181/18/18_appendix.pdf. *Source code* lengkap *benchmark* program yang digunakan untuk data masukan (*input*). Berikut *pseudocode* program klasifikasi *Myers Triangel*.

```
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";
```

3.3 *Parser* (*Parsing*)

Parser adalah proses menganalisis serangkaian symbol dalam suatu bahasa, baik bahasa alami, bahasa komputer atau struktur data. Dalam ilmu komputer, penguraian atau *parsing* adalah suatu cara memecah-mecah suatu rangkaian *input*

yang akan menghasilkan suatu pohon uraian (*parse tree*) penguraian dapat didahului atau diikuti oleh langkah-langkah lain, atau dapat digabungkan menjadi satu langkah (T Manikumar dkk, 2016).

```
static double myerstriangle(double A, double B, double C){
double type = 0;
    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 = 1; //"equilateral"
                } else {
                    type = 2; //"isosceles"
                }
            } else if (A == C) {
                type = 2; //"isosceles"
            } else if (B == C) {
                type = 2; //"isosceles"
            } else {
                type = 3; //"scalene"
            }
        } else {
            type = 4; //"notriangle"
        }
    } else {
        type = 4; //"notriangle"
    }
    return type;
}
}
```

Gambar 3. 2 Hasil *Parsing Source Code Myers Triangel*

Pada tahap *parsing* program, proses yang dilakukan adalah data uji *myers triangle* di *parsing* dimana *sintaks-sintaks* dari *query* akan dicek untuk menentukan agar *query* tersebut telah sesuai dengan ketentuan *sintaks* dari bahasa *query*. Berikut *source code* untuk *parsing* data uji.

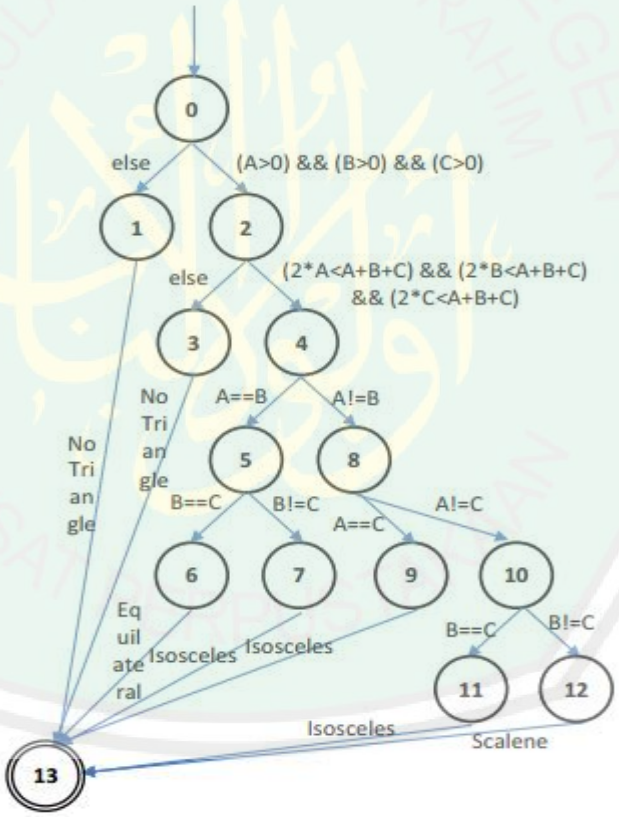
```
public Parser(String path) throws IOException {
            ArrayList<String> lines = new
ArrayList<>();
            FileReader fileReader
                    = new FileReader(path);
            BufferedReader.
            BufferedReader bufferedReader
                    = new
BufferedReader(fileReader);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

3.4 *Control Flow Graph* (*CFG*)

Control Flow Graph (*CFG*) adalah representasi grafis dari aliran kontrol atau perhitungan selama eksekusi program atau aplikasi. Grafik aliran kontrol sebagian besar digunakan dalam analisis statis serta aplikasi kompiler, karena grafik tersebut dapat secara akurat mewakili aliran di dalam unit program. Grafik aliran kontrol pada awalnya dikembangkan oleh Frances E. Allen.CGF. Setiap *Control Flow Graph* (*CFG*) terdiri *edges* dan *nodes*. *Edges* mempresentasikan kontrol transfer antar *nodes,* sedangkan *nodes* mempresentasikan perintah (Tracey, N, 1998).

Sebagai contoh, pertimbangkan kode sumber yang ditunjukkan pada Gambar 3.2 untuk masalah klasifikasi segitiga. Fungsi ini menerima panjang tiga sisi segitiga dan memeriksa apakah dapat membentuk segitiga atau tidak. Jika

demikian, ia mengembalikan jenis segitiga, jika tidak mencetaknya bukan segitiga. Pada langkah pertama, kode sumber diurai untuk menemukan pernyataan percabangan yang tersedia untuk setiap cabang, sebuah simpul dibuat untuk *Control Flow Graph* (*CFG*) dengan dua jalur berbeda, untuk masing-masing kasus "benar" dan "salah". Level grafik berikutnya berlanjut jika ada cabang bersarang yang tersedia. Simpul nonleaf diidentifikasi sebagai simpul target. Akhirnya, semua simpul nonleaf terhubung ke simpul "keluar". *Control Flow Graph* (*CFG*) yang sesuai untuk kode sumber tersebut ditunjukkan pada Gambar 3.3.



Gambar 3. 3 *Control Flow Graph* (*CFG*) untuk *klasifikasi segitiga Myers* (Sumber : T Manikumar dkk, 2016).

Setelah melalui proses anilisis *Control Flow Graph* (*CFG*) yang menghasilkan gambara aliran *path* kode program. Berikut merupakan semua kemungkinan jalur atau *path* yang dapat dijadikan sebagai dasar dalam pembangkitan data uji, kemungkinan jalur atau *path* dipresentasikan dalam bentuk urutan huruf *node*. Berikut *source code Control Flow Graph* (*CFG*) yang akan membentuk jalur *node* yang benar.

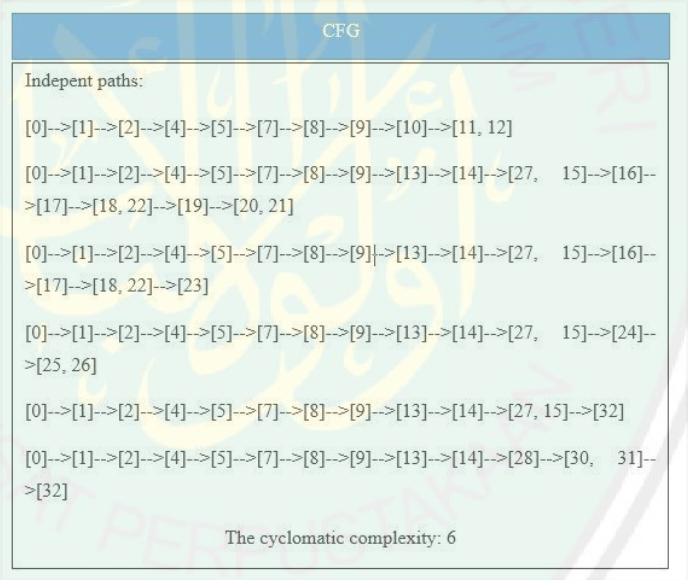
```
public Node buildGraph() {
            Node root = new Node(nodeName, 0);
            nodeName++;
            buildChild(root, true);

            ArrayList<Node> 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;
            text_cfg.append("Indepent paths:\n");
            cyclomaticComplexity = findPaths(root,
"", cyclomaticComplexity);

            text_cfg.append("\n");
            text_cfg.append("The cyclomatic
complexity: " + cyclomaticComplexity);
            return root;
        }

        private int findPaths(Node root, String
string, int cyclomaticComplexity) {
            boolean isLoop = false;
            if
(string.contains(String.valueOf(root.lines))) {
                isLoop = true;
            }
            string =
string.concat(String.valueOf(root.lines));
            if (root.childs.isEmpty()) {
                text_cfg.append(string + "\n");
                cyclomaticComplexity++;
            } else {
                string = string.concat("-->");
            }
            ArrayList<Node> childs = root.childs;
            Node current;

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

3.5 *Path* (Jalur)

Path adalah sebuah jalur pada program yang mengandung paling sedikit sebuah pernyataan lojik (Pressman, 2010). Dalam konteks grafik, maka sebuah jalur disebut sebagai jalur dasar jika jalur tersebut memiliki paling tidak satu buah simpul yang belum pernah dilalui oleh jalur yang sudah didefinisikan sebelumnya.

Masukan (*input*) berupa fungsi dari program kemudian dianalisis menggunakan *control flow graph* (*CFG*) yang menggambarkan aliran (*path*) kode program.

Dari grafik alur pada Gambar 3.3 didapatkan jumlah *path* jalur program dalam bentuk urutan nomer *node* sebagai gambar berikut :



Gambar 3. 4 Jalur *Path*

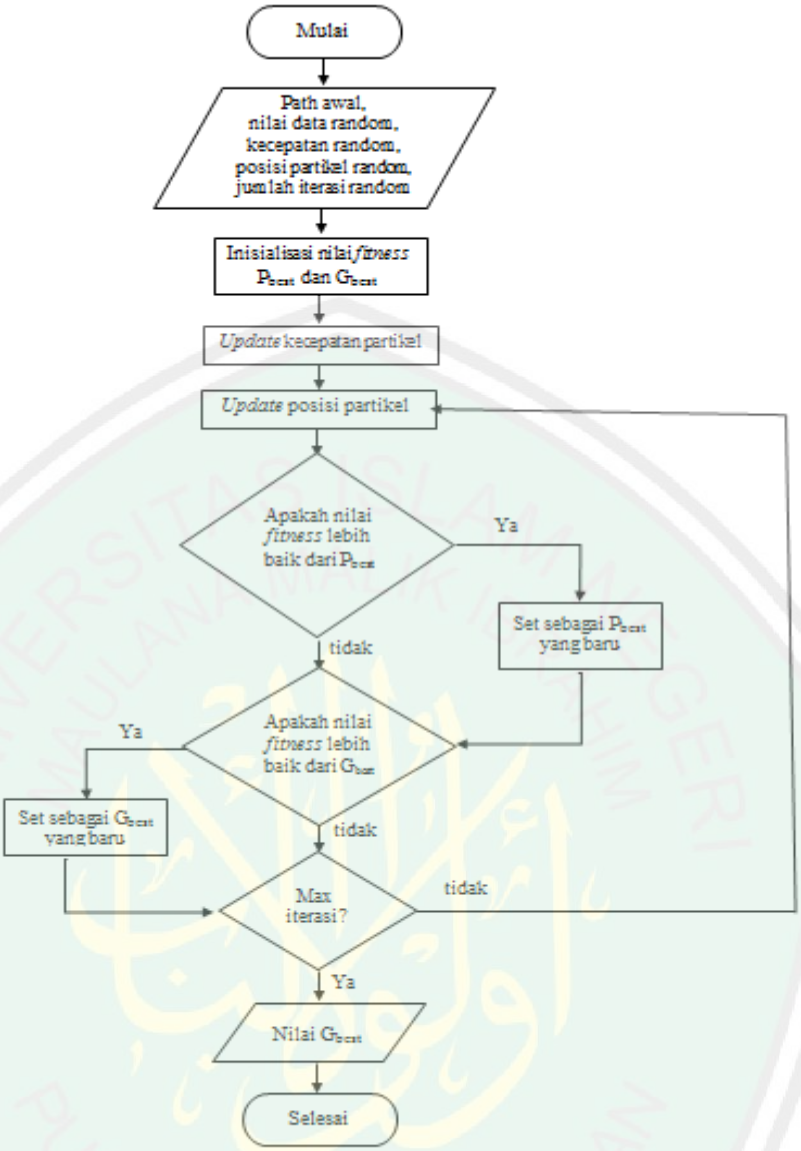
3.6 Metode *Particle Swarm Optimization*

Algoritma *Particle Swarm Optimization* adalah algoritma optimasi yang sangat mudah untuk di pahami dan dimengerti, karna cukup sederhana dan memiliki kinerja yang terbukti sangat bagus. Algoritma *Particle Swarm*

Optimization dapat berguna di berbagai masalah optimasi baik kontinyu maupun distrik, linier maupun nonlinier (yuli, 2013).

Particle Swarm Optimization memodelkan aktivitas pencarian solusi terbaik dalam suatu ruang solusi sebagai aktivitas terbangnya kelompok partikel dalam suatu ruang solusi tersebut. Posisi partikel dalam ruang solusi tersebut merupakan kandidat solusi yang berisi variabel-variabel optimasi. Setiap posisi tersebut akan dikaitkan dengan sebuah nilai yang disebut nilai objektif atau nilai *fitness* yang dihitung berdasarkan fungsi objektif dari masalah optimasi yang akan diselesaikan.

Pada penelitian ini dibangkitkan data uji atau data *test* dengan menginialisasi nilai data secara *random*, kecepatan partikel, posisi partikel dan jumlah iterasi 1, kemudian mencari nilai *fitness*, *Pbest* dan *Gbest*nya sampai menemukan hasil yang konvergen.



Gambar 3. 5 *Flowchart Particle Swarm Optimization*

Pendeskripsian solusi pada algortima *Particle Swarm Optimization* yang menghasilkan baik atau jelek menggunakan nilai objektif. Masing-masing solusi dihitung nilai objektifnya. Nilai objektif tersebut adalah keluaran dari fungsi objektif. Nilai objektif yang menggambarkan *fitness* sebuah solusi merupakan nilai *fitness*, sehingga fungsi objektif juga merupakan fungsi *fitness*.

Pada kasus pembangkitan kasus data uji yang menggunakan fungsi objektif adalah fungsi *cost* usulan dari (Bottaci, L 2003). Fungsi *cost* merupakan fungsi pinalty yang merepresentasikan jarak cabang (*branch distance*) yang akan di hitung jika predikat cabang tersebut tidak terpenuhi. Fungsi *cost* terhadap pemenuhan ekspresi predikat dengan a, b adalah bilangan riil dan k adalah bilangan riil positif, berikut ditunjukan pada tabel 3.1. "Predikat A AND Predikat B" "(A&&B)" dan "Predikat A OR Predikat B" "(A||B)" adalah ekpresi predikat operasi logika.

Tabel 3. 1 Fungsi *Cost* atau *Fitness* (Sumber : Mandyartha, 2017)

No	Ekspresi Predikat	Cost bila tidak memenuhi Ekpresi Predikat
1	$a \leq b$	$a - b$
2	$a < b$	$a - b + k$
3	$a = b$	$abs\ (a - b)$
4	$a \neq b$	$k - abs\ (a - b)$
5	$a \geq b$	$b - a$
6	$a > b$	$b - a + k$
7	$A\ \&\&\ B$	$\max(\cos(A), cost(B))$
8	$A\ \|\|\ B$	$\min(\cos(A), cost(B))$

Pada proses ini menjelaskan bagaimana cara untuk membangkitkan data *test* secara manual menggunakan algoritma *Particle Swarm Optimization* dengan

cara pada Tabel 3.1. Berikut langkah-langkah untuk membangkitkan data *test* secara manual

1. Pemilihan *path* secara random, *path* yang dipilih adalah *path* yang terbentuk pada langkah 3.5 yaitu *path* (jalur). Pada tahap ini saya memilih *path* ke 2 untuk membangkitkan data *test*.

   Path 2 = 0 – 2 – 3 – 13

2. Menentukan nilai data random dari kode program *benchmark* A, B, dan C untuk menghasilkan nilai *fitness* atau nilai objektif.

   Nilai random = nilai A:4, nilai B:6, dan nilai C:7

3. Menghitung jumlah *fitness* untuk kode program pada gambar 3.3 yang telah terbentuk menjadi *Control Flow Graph* dengan menggunakan menggunakan nilai cost pada Tabel 3.1.

   - Pengujian path 0 – 2 – 3 – 13
   - Nilai data random A = 4 B = 6 C = 7
   - Ekpresi predikat yang harus di eksekusi A>0 && B>0 && C>0, di invers menjadi 2*A≥A+B+C || 2*B≥A+B+C || 2*C≥A+B+C karena harus melalui node 2 yang bernilai false.
   - Kondisi 2*4≥4+6+7 = 8≥17 maka A bernilai *false*, sehingga dihitung fungsi costnya 17-8 = 9
   - Kondisi 2*6≥4+6+7 = 12≥17 maka B bernilai *false*, sehingga dihitung fungsi costnya 17-12 = 5
   - Kondisi 2*7≥4+6+7 = 14≥17 maka C bernilai *false*, sehingga dihitung fungsi costnya 17-14 = 3

- Total nilai *fitness* awal pada ekpresi predikat A || B || C sehingga (4,6,7) adalah 3.

4. Kemudian memasukan angka dengan menggunakan proses algoritma *Particle Swarm Optimization.*

   a) Menetapkan jumlah partikel *(N)* sebagai data uji. Sebagai contoh ditetapkan N=3. Contoh data uji adalah sebagai berikut.

      $x1(0) = 4,$

      $x2(0) = 6,$

      $x3(0) = 7$

   b) Melakukan evaluasi data uji terhadap solusi atau persamaan pengujian. Dengan fungsi *fitness* (2*A ≥ A+B+C) ‖ (2*B ≥ A+B+C) ‖ (2*C ≥ A+B+C)

      $f1 = f(4) \quad = 3$

      $f2 = f(6) \quad = 3$

      $f3 = f(7) \quad = 3$

   c) Menentukan kecepatan awal pada penelitian ini adalah

      v1(0) = 0,

      v2(0) = 0,

      v3(0) = 0,

   d) Menentukan iterasi awal = 1

   e) Menentukan *Pbest* Terakhir

      *Pbest*1(0) = 4,

      *Pbest*2(0) = 6,

      *Pbest*3(0) = 7,

Gbest = 3

f) Menghitung perubahan nilai kecepatan dengan cara membangkitkan nilai Hitung v(j) dengan c1 = c2 = 1. Misalkan nilai *random* yang didapat, r1 =0 .4,r2 =0 .5, dengan rumus

$$Vj(i) = Vj(I-1) + c1r1[Pbest,j - xj(i-1)] + c2r2[Gbest - xj(i-1)]$$

Diperoleh

$$v1(1) = 0 + 0.4(4-4) + 0.5(3-4) = -0.5$$

$$v2(1) = 0 + 0.4(6-6) + 0.5(3-6) = -1.5$$

$$v3(1) = 0 + 0.4(7-7) + 0.5(3-7) = -2$$

Sedangkan untuk nilai *x* dengan rumus $Xj(i)=Xj(i-1)+Vj(i)$ adalah

$$x1(1) = 4 + (-0.5) = 3.5$$

$$x2(1) = 6 + (-1.5) = 4.5$$

$$x3(1) = 7 + (-2) = 5$$

g) Melakukan evaluasi data uji berdasarkan hasil perhitungan perubahan kecepatan dengan rumus $f(x) = (Gbest - x)^2$

$$f1 = (3.5) = (3-3.5)^2 = -0.5^2 = 0.25$$

$$f2 = (4.5) = (3-4.5)^2 = -1.5^2 = 2.25$$

$$f3 = (5) = (3-5)^2 = -2^2 = 4$$

Sedangkan pada iterasi sebelumnya kita dapatkan

$$f1(0) = f(4) = 3,$$

$$f2(0) = f(6) = 3,$$

$$f3(0) = f(7) = 3,$$

Nilai dari *f* dari iterasi sebelumnya tidak ada yang lebih baik sehingga *Pbest* untuk masing-masing partikel sama dengan nilai *x*-nya.

Gbest = 3.

h) Dari hasil analisa didapatkan nilai *gbest* terakhir juga tidak ada perubahan yaitu *Gbest* = 3

i) Cek apakah solusi x sudah konvergen, dimana nilai x saling dekat. Jika tidak, tingkatkan ke iterasi berikutnya i = 2. Lanjut ke langkah e.

j) Hasil *Pbest* baru

*Pbest*1 = 3.5,

*Pbest*2 = 4.5,

*Pbest*3 = 5,

k) Hitung kecepatan baru dengan bilangan *random* r1=0.3, r2-0.6. Hasil dari perhitungan kecepatan baru adalah sebagai berikut ini.

$v1(2) = -0.5 + 0.3(3.5 - 3.5) + 0.6(3 - 3.5) = -0.8$

$v2(2) = -1.5 + 0.3(4.5 - 4.5) + 0.6(3 - 4.5) = -2.4$

$v3(2) = -2 + 0.3(5 - 5) + 0.6(3 - 5) = -3.2$

Sedangkan untuk nilai x adalah

$x1(2) = 3.5 + (-0.8) = 2.7$

$x2(2) = 4.5 + (-2.4) = 2.1$

$x3(2) = 5 + (-3.2) = 1.8$

Melakukan evaluasi data uji berdasarkan hasil perhitungan perubahan kecepatan di iterasi ke 2

$f1(2) = f(3.5) = (3 - 2.7)^2 = 0.3^2 = 0.09$

$f2(2) = f(4.5) = (3 - 2.1)^2 = 0.9^2 = 0.81$

$f3(2) = f(1.8) = (3 - 1.8)^2 = 1.2^2 = 1.44$

Jika dibandingkan dengan nilai *f* dari iterasi sebelumnya, ada nilai yang lebih baik dari nilai *f* sekarang yaitu $f2(3.5) = 0.09$, sehingga *Pbest* yang lebih baik yaitu 3.5 pada iterasi ke 2 dan *Gbest* = 3.

Cek apakah solusi sudah konvergen, dimana nilai *Gbest* tidak mengalami perubahan. Maka iterasi dihentikan, jika *Gbest* mengalami perubahan maka dilanjutkan iterasi selanjutnya dengan menghitung kecepatan dan ulang langkah-langkah selanjutnya, sampai mencapai nilai konvergen.

Dari hasil perhitungan algoritma *Particle Swarm Optimization* ditunjukan bahwa nilai data *test* di ambil dari nilai *Pbest* yang baru, jika nilai tersebut belum bisa mencapai nilai konvergen maka iterasi di lanjut sampai menemukan hasil yang konvergen dengan nilai *fitness* atau nilai *Gbest* yang baik.

3.7 Pengujian algoritma *Particle Swarm Optimization* Secara Otomatis

Proses algoritma *Particle Swarm Optimization* secara otomatis maka hasilnya akan terlihat lebih detail dan lebih tepat sesuai dengan perhitungan algoritma *Particle Swarm Optimization*. Pada tahap penerapan metode *Particle Swarm Optimization*, proses ini dilakukan ketika hasil *path* atau jalur dari proses *Control Flow Graph* (CFG) telah berhasil terbentuk. Kemudian metode *Particle Swarm Optimization* ini bekerja dengan menguji masing-masing *path* atau jalur, untuk mengetahui *path* atau jalur mana yang dilalui data *test*. Berikut *source code* penerapan metode *Particle Swarm Optimization*.

```
//PSo implementationoos
    public void menu(boolean flag) {
        Swarm swarm;
        Particle.FunctionType function;
        int particles, epochs;
        double inertia, cognitive, social;

        function = getFunction();
        particles = 10;
        epochs = 1;

        if (flag) {
            inertia = getUserDouble("Inertia:   ");
            cognitive = getUserDouble("Cognitive: ");
            social = getUserDouble("Social:    ");
            swarm = new Swarm(function, particles,
epochs, inertia, cognitive, social);
        } else {
            swarm = new Swarm(function, particles,
epochs);

        }

        swarm.run();
        text_output.append("X: " + Particle.x + ", ");
        text_output.append("Y: " + Particle.y + ", ");
        text_output.append("Z: " + Particle.z + ", ");
        text_output.append("\n");
        text_output.append("-------------------
EXECUTING-----------------" + "\n");
        text_output.append("Global Best Evaluation
(Epoch " + 0 + "):\t" + swarm.bestEval + "\n");
        text_output.append("Final Best Evaluation: " +
swarm.bestEval + "\n");
        text_output.append("-------------------
COMPLETE-----------------");

    }

    private static Particle.FunctionType getFunction()
{
        Particle.FunctionType function = null;

        return function;
    }
```

```
private static int getUserInt(String msg) {
        int input;
        while (true) {
            Scanner sc = new Scanner(System.in);
            System.out.print(msg);

            if (sc.hasNextInt()) {
                input = sc.nextInt();

                if (input <= 0) {
                    System.out.println("Number must
be positive.");
                } else {
                    break;
                }

            } else {
                System.out.println("Invalid
input.");
            }
        }
        return input;
    }

    private static double getUserDouble(String msg)
{
        double input;
        while (true) {
            Scanner sc = new Scanner(System.in);
            System.out.print(msg);

            if (sc.hasNextDouble()) {
                input = sc.nextDouble();

                if (input <= 0) {
                    System.out.println("Number must
be positive.");
                } else {
                    break;
                }

            } else {
                System.out.println("Invalid
input.");
            }
        }
        return input;
    }
```

BAB IV

UJI COBA DAN PEMBAHASAN

Dalam bab ini membahas mengenai hasil uji serta pembahasan bagaimana proses jalannya aplikasi, uji coba dilakukan untuk mengetahui apakah aplikasi dapat berjalan sesuai dengan apa yang diharapkan.

4.1. Hasil

Pengujian adalah bagian terpenting dalam proses pembuatan aplikasi. Pengujian juga dilakukan untuk menjamin kualitas aplikasi yang akan dibuat dan mengetahui kelemahan dari aplikasi yang dibuat. Kasus uji yang baik yaitu yang memiliki tingkat kemungkinan tinggi untuk kerusakan yang belum ditentukan.

4.1.1 Data Uji Coba

Inputan untuk pembakitan data test pada pengujian perangkat lunak menggunakan data *test myers triangle*, data *test myers triangle* juga digunakan oleh penelitian lainnya seperti pachauri et al, dan ferrer et al. Data *test myers triangle* menggolongkan segitiga dasar sisi *input* sebagai non-segitiga, yaitu, sama kaki, sama sisi atau sisi tak sama Panjang. Diberikan tiga *input* nilai di dalam parameter A, B dan C yang semuanya mewakili sisi segitiga. Berikut adalah data *test myers triangle* :

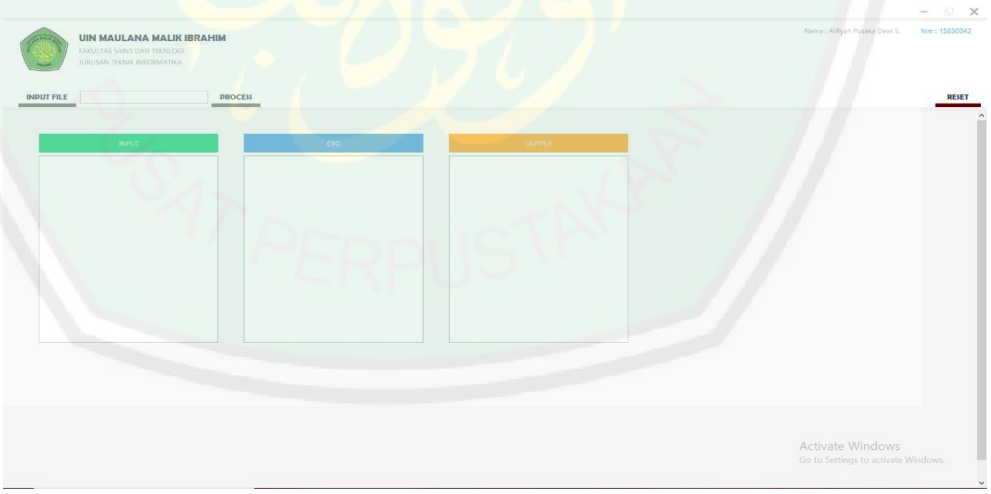
```
public static void Triangle(double A, double B, double
C){
String type="";
double perimeter;
if (A > 0 && B > 0 && C > 0){
perimeter = A + B + C;
if (((2 * A) < perimeter) && ((2 * B) < perimeter)
&&
((2 * C) < perimeter)){
if (A == B) {
if (B == C) {
type = "equilateral";
}else{
type = "isosceles";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

Data *test sthamer triangle* juga sama dengan data *test myers triangle* yang menggolongkan segitiga atas dasar sisi input sebagai non-segitiga atau segitiga, yaitu, sama kaki, sama sisi, segitiga sudut kanan, atau sisi tak sama panjang. Berikut adalah data *test sthamer triangle* :

```
static int triangle3(double a, double b, double c) {
double perim;
int tri_kind; //0-no triangle, 1-isoscele, 2-
equilateral, 3-right angle triangle, 4-triangle
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) {
/**
* *** triangle2 ****
*/
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;
} else /**
* *** right_angle_check ****
*/
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;
}
```

4.1.2. Proses Uji Coba Aplikasi

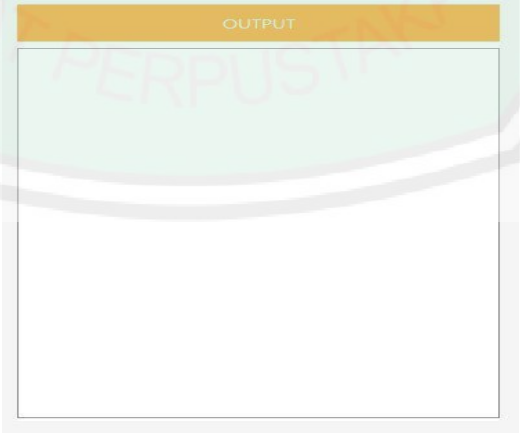
Aplikasi pembangkitan data *test* ini di rancang menggunakan Bahasa pemrograman java. Pembuatan aplikasi ini menggunakan *Netbeans IDE 8.0* sebagai *text* editor untuk penulisan kode program. Untuk mempermudah pengguna dalam mengoperasikan aplikasi ini dibuat sebuah tampilan atau *interface* berbasis *Grapic User Interface* (GUI) yang akan dijelaskan lebih rinci sebagai berikut. Pertama kali aplikasi dijalankan adalah menampilkan halaman utama yang berisikan menu-menu aplikasi seperti Gambar 3.10, terdapat menu *input* untuk menginutkan data *test* yang telah di parsing, kemudian terdapat *Control Flow Graph* (*CFG*) yang menunjukkan hasil dari jalur *path* atau *node*, kemudian menunjukan hasil output parameter A, B dan C yang sesuai dengan perhitungan Algoritma *Particle Swarm Optimation*.



Gambar 4. 1 Tampilan Halaman Utama Aplikasi

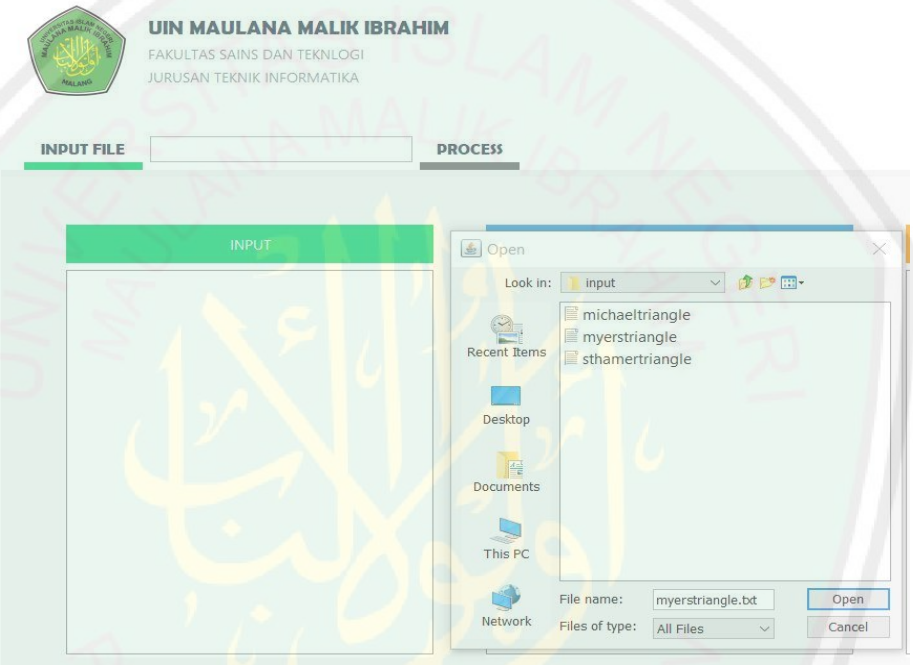
Gambar 4. 2 Tampilan Kolom *Input*

Gambar 4. 3 Tampilan Kolom *control flow graph* (*CFG*)



Gambar 4. 4 Tampilan Kolom *Output* atau Hasil

Pada tampilan utama pada Gambar 4.1 menampilkan 3 kolom yaitu, kolom *input* yang digunakan untuk memasukan data *test* ditunjukkan pada Gambar 4.2, kemudian kolom *control flow graph* (*CFG*) yang akan menampilkan hasil *parsing* dari data *test* yang membentuk suatu jalur atau *node* yang ditunjukkan pada gambar 4.3, lalu kolom terakhir adalah *output* atau hasil dari data *test* yang diuji melalui metode *Particle Swarm Optimation* yang ditunjukkan pada gambar 4.4.



Gambar 4. 5 Pemilihan Data *Test*

INPUT

```
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";
```

Gambar 4. 6 *Input* data *test*

```
type = "isosceles";
}else{
type = "scalene";
}
}
}
}else{
type = "notriangle";
}
}else{
type = "notriangle";
}
}
```

Gambar 4. 7 *Input* data *test* lanjutan

Pengujian pertama kali dilakukan dengan cara klik tombol *input file*, setelah itu akan muncul beberapa data *test* pada *direktori* komputer, kemudian kita dapat memilih data *test* yang akan diuji. Pada pengujian ini data yang dipilih adalah *myers triangle*. Setelah data dipilih, data akan menunjukan hasil data *test*

myers triangle pada Gambar 4.6 dan 4.7, setelah gambar *input* berhasil keluar kemudian hasil *input* di proses dengan *parsing* agar membentuk *Control Flow Graph* (*CFG*) pada Gambar 4. 8 dan 4. 9.

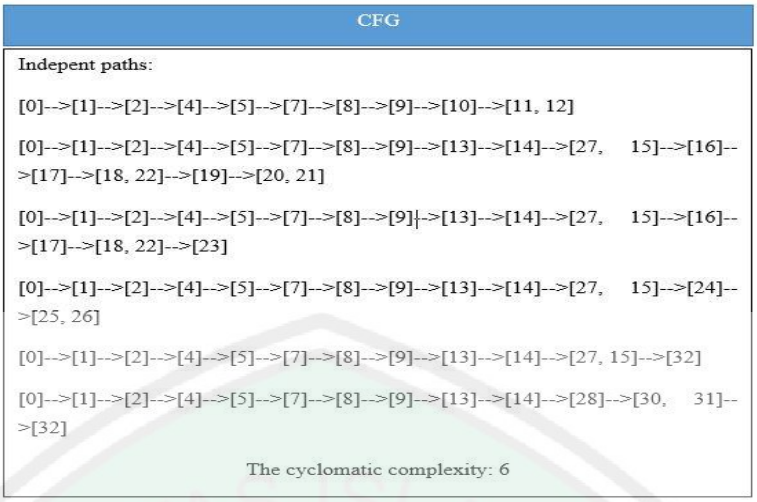
Gambar 4. 8 Hasil *parsing* data t*est*

OUTPUT

```
16:}else{
17:if (B == C){
18:type = "isosceles";
19:}else{
20:type = "scalene";
21:}
22:}
23:}
24:}else{
25:type = "notriangle";
26:}
27:}
28:else
29:{
30:type = "notriangle";
31:}
32:}
```

Gambar 4. 8 Hasil *parsing* data *test* lanjutan

Setelah berhasil menampilkan data *test* yang sudah *terparsing*, data ***test*** *myers triangle* membentuk *Control Flow Graph* (*CFG*) yang menghasilkan jalur *path* atau *node* untuk mengetahui jalur manakah yang benar untuk dilalui pengujian metode *Particle Swarm Optimation* yang ditunjukkan pada Gambar 4.10.



Gambar 4. 9 Hasil jalur *control flow graph* (*CFG*)

4.2. Pembahasan Hasil Uji Coba

Berdasarkan hasil uji coba yang telah dilakukan, aplikasi berhasil mendapatkan *variabel* untuk penyelesaian masalah dari data *test myers triangle* dan *sthamer triangel* dengan menggunakan metode *Particle Swarm Optimation.* Hal ini membuktikan bahwa metode *Particle Swarm Optimation* berhasil membangkitkan nilai *variabel* data *test* secara otomatis dengan tepat.

4.2.1 Pengujian dan Hasil pada data *test myers triangle*

Pengujian data *test myers triangle* didapatkan hasil jalur *path* yang terbentuk dari *control flow graph* (*CFG*) dan hasil pembangkitan data *test* dari *path* tersebut diproses menggunakan metode *Particle Swarm Optimation. Control flow graph* (*CFG*) yang dihasilkan memiliki 6 jalur *independent* atau jalur *path* pada Gambar 4.11.

CFG

Indepent paths:

[0]-->[1]-->[2]-->[4]-->[5]-->[7]-->[8]-->[9]-->[10]-->[11, 12]

[0]-->[1]-->[2]-->[4]-->[5]-->[7]-->[8]-->[9]-->[13]-->[14]-->[27, 15]-->[16]-->[17]-->[18, 22]-->[19]-->[20, 21]

[0]-->[1]-->[2]-->[4]-->[5]-->[7]-->[8]-->[9]-->[13]-->[14]-->[27, 15]-->[16]-->[17]-->[18, 22]-->[23]

[0]-->[1]-->[2]-->[4]-->[5]-->[7]-->[8]-->[9]-->[13]-->[14]-->[27, 15]-->[24]-->[25, 26]

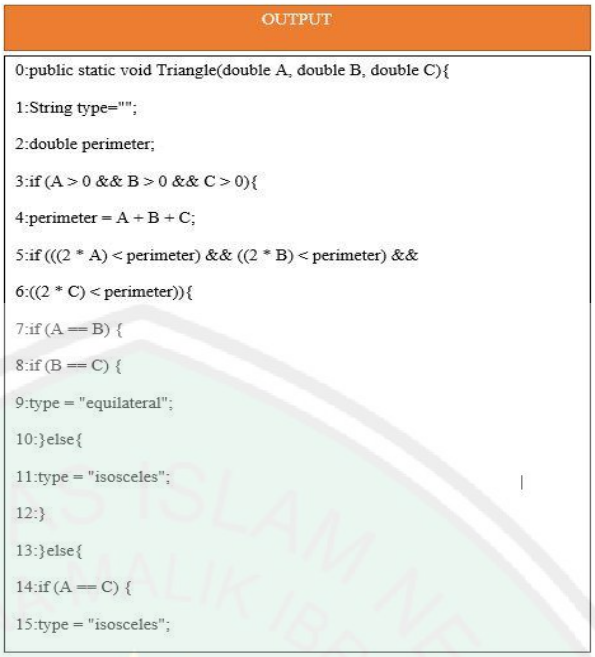
[0]-->[1]-->[2]-->[4]-->[5]-->[7]-->[8]-->[9]-->[13]-->[14]-->[27, 15]-->[32]

[0]-->[1]-->[2]-->[4]-->[5]-->[7]-->[8]-->[9]-->[13]-->[14]-->[28]-->[30, 31]-->[32]

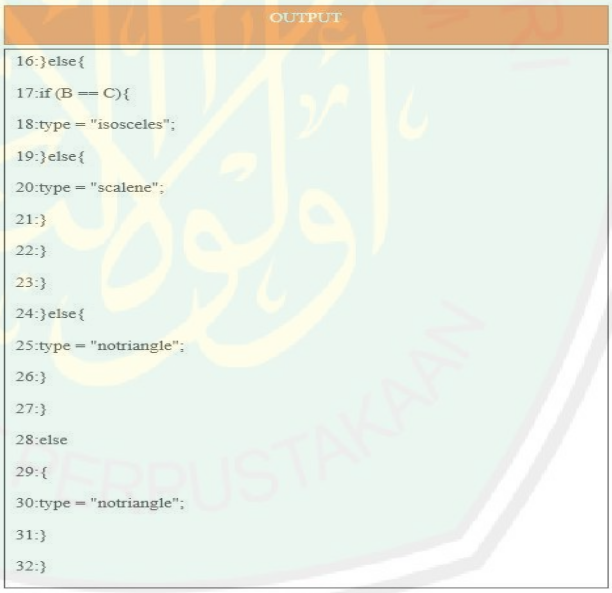
The cyclomatic complexity: 6

Gambar 4. 10 *Control flow graph* (*CFG*) data *test myers triangle*

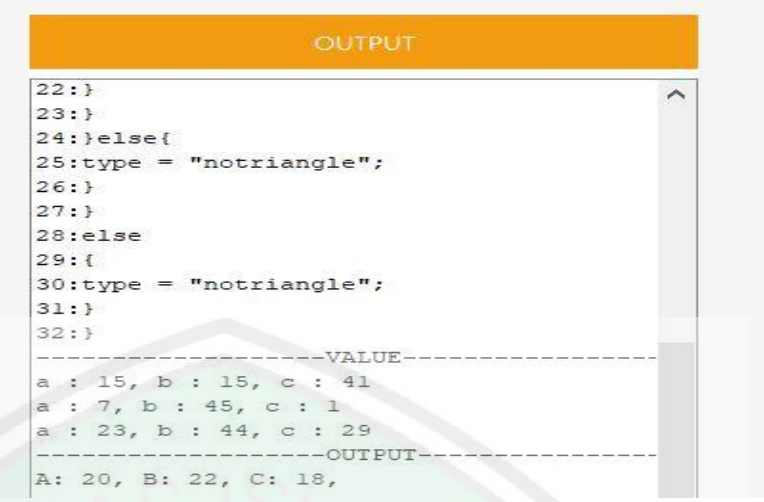
Setelah *Control flow graph* (*CFG*) terbentuk kemudian lanjut dengan pengujian data *test* menggunakan metode *Particle Swarm Optimation* untuk menemukan nilai hasil *variable* dari data *test myers triangle*.



Gambar 4. 11 Hasil data *test myers triangle*



Gambar 4. 13 Hasil data *test myers triangle* lanjutan



Gambar 4. 14 Hasil dan *output* data *test myers triangle*

Tabel 4. 1 *Validasi* hasil *Myers Triangle*

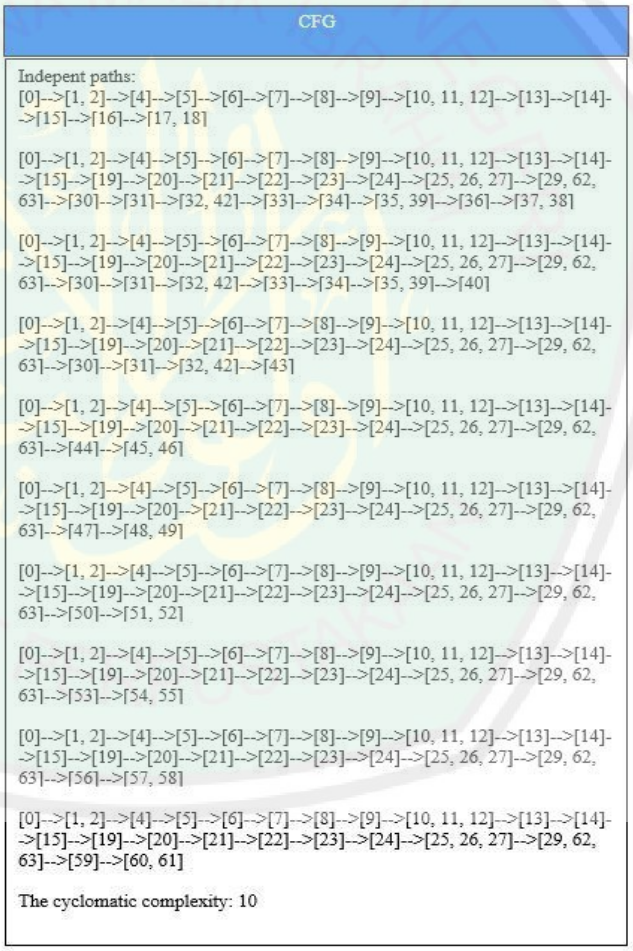
No	Hasil data *test myers triangle*			*Kualifikasi* hasil
1	A	B	C	
2	15	15	41	Bukan segitiga
3	7	45	1	Bukan segitiga
4	23	44	29	Segitiga tak sama Panjang
5	20	22	18	Segitiga tak sama panjang

Pada Gambar 4. 12, 4. 13 dan 4. 14 menunjukkan hasil data *test myers triangle* yang *konvergen* dengan perhitungan metode *Particle Swarm Optimation*. Pada tabel 4. 1 menunjukkan pengujian *kualifikasi* hasil dari dari beberapa hasil *variable* data *test myers triangle.* Data *test* A = 20, B = 22 dan C = 18 adalah hasil data *test myers triangle* yang terbaik dari beberapa hasil data *test* yang ada, karena hasil data *test* tersebut

menunjukkan *kualifikasi* segitiga tak sama panjang dengan angka yang paling kecil diantara yang lainnya.

4.2.2 Pengujian dan Hasil pada data *test sthamer triangle*

Pengujian data *test sthamer triangle* didapatkan hasil jalur path yang terbentuk dari *control flow graph (CFG)* dan hasil pembangkitan data test dari path tersebut diproses menggunakan metode *Particle Swarm Optimation*. *Control flow graph (CFG)* yang dihasilkan memiliki 10 jalur *independent* atau jalur *path* pada Gambar 4.15.


CFG

Indepent paths:
[0]-->[1, 2]-->[4]-->[5]-->[6]-->[7]-->[8]-->[9]-->[10, 11, 12]-->[13]-->[14]-->[15]-->[16]-->[17, 18]

[0]-->[1, 2]-->[4]-->[5]-->[6]-->[7]-->[8]-->[9]-->[10, 11, 12]-->[13]-->[14]-->[15]-->[19]-->[20]-->[21]-->[22]-->[23]-->[24]-->[25, 26, 27]-->[29, 62, 63]-->[30]-->[31]-->[32, 42]-->[33]-->[34]-->[35, 39]-->[36]-->[37, 38]

[0]-->[1, 2]-->[4]-->[5]-->[6]-->[7]-->[8]-->[9]-->[10, 11, 12]-->[13]-->[14]-->[15]-->[19]-->[20]-->[21]-->[22]-->[23]-->[24]-->[25, 26, 27]-->[29, 62, 63]-->[30]-->[31]-->[32, 42]-->[33]-->[34]-->[35, 39]-->[40]

[0]-->[1, 2]-->[4]-->[5]-->[6]-->[7]-->[8]-->[9]-->[10, 11, 12]-->[13]-->[14]-->[15]-->[19]-->[20]-->[21]-->[22]-->[23]-->[24]-->[25, 26, 27]-->[29, 62, 63]-->[30]-->[31]-->[32, 42]-->[43]

[0]-->[1, 2]-->[4]-->[5]-->[6]-->[7]-->[8]-->[9]-->[10, 11, 12]-->[13]-->[14]-->[15]-->[19]-->[20]-->[21]-->[22]-->[23]-->[24]-->[25, 26, 27]-->[29, 62, 63]-->[44]-->[45, 46]

[0]-->[1, 2]-->[4]-->[5]-->[6]-->[7]-->[8]-->[9]-->[10, 11, 12]-->[13]-->[14]-->[15]-->[19]-->[20]-->[21]-->[22]-->[23]-->[24]-->[25, 26, 27]-->[29, 62, 63]-->[47]-->[48, 49]

[0]-->[1, 2]-->[4]-->[5]-->[6]-->[7]-->[8]-->[9]-->[10, 11, 12]-->[13]-->[14]-->[15]-->[19]-->[20]-->[21]-->[22]-->[23]-->[24]-->[25, 26, 27]-->[29, 62, 63]-->[50]-->[51, 52]

[0]-->[1, 2]-->[4]-->[5]-->[6]-->[7]-->[8]-->[9]-->[10, 11, 12]-->[13]-->[14]-->[15]-->[19]-->[20]-->[21]-->[22]-->[23]-->[24]-->[25, 26, 27]-->[29, 62, 63]-->[53]-->[54, 55]

[0]-->[1, 2]-->[4]-->[5]-->[6]-->[7]-->[8]-->[9]-->[10, 11, 12]-->[13]-->[14]-->[15]-->[19]-->[20]-->[21]-->[22]-->[23]-->[24]-->[25, 26, 27]-->[29, 62, 63]-->[56]-->[57, 58]

[0]-->[1, 2]-->[4]-->[5]-->[6]-->[7]-->[8]-->[9]-->[10, 11, 12]-->[13]-->[14]-->[15]-->[19]-->[20]-->[21]-->[22]-->[23]-->[24]-->[25, 26, 27]-->[29, 62, 63]-->[59]-->[60, 61]

The cyclomatic complexity: 10

Gambar 4. 15 *Control flow graph (CFG) data test sthamer triangle*

Setelah *Control flow graph* (*CFG*) terbentuk kemudian lanjut dengan pengujian data *test* menggunakan metode *Particle Swarm Optimation* untuk menemukan nilai hasil *variable* dari data *test sthamer triangle*.

Gambar 4. 16 Hasil data *test sthamer triangle*

OUTPUT

```
19:} else {
20:if (a == c) {
21:tri_kind = 1;
22:} else {
23:if (b == c) {
24:tri_kind = 1;
25:} else /**
26:* *** right_angle_check ****
27:*/
28:if ((a * a + b * b) == (c * c)) {
29:tri_kind = 3;
30:} else {
31:if ((b * b + c * c) == (a * a)) {
32:tri_kind = 3;
33:} else {
34:if ((a * a + c * c) == (b * b)) {
35:tri_kind = 3;
36:} else {
37:tri_kind = 4;
38:}
39:}
```

Gambar 4. 17 Hasil data *test sthamer triangle* lanjutan

OUTPUT

```
40:}
41:
42:}
43:}
44:} else {
45:tri_kind = 0;
46:}
47:} else {
48:tri_kind = 0;
49:}
50:} else {
51:tri_kind = 0;
52:}
53:} else {
54:tri_kind = 0;
55:}
56:} else {
57:tri_kind = 0;
58:}
59:} else {
60:tri_kind = 0;
```

Gambar 4. 18 *Hasil data test sthamer triangle* lanjutan

OUTPUT

```
53:                 } else {
54:                     tri_kind = 0;
55:                 }
56:             } else {
57:                 tri_kind = 0;
58:             }
59:         } else {
60:             tri_kind = 0;
61:         }
62:         return tri_kind;
63:     }
--------------------VALUE------------------
a : 11, b : 34, c : 33
a : 0, b : 8, c : 19
a : 19, b : 27, c : 0
--------------------OUTPUT-----------------
A: 33, B: 38, C: 7,
```

Gambar 4. 19 Hasil dan output data *test sthamer triangle*

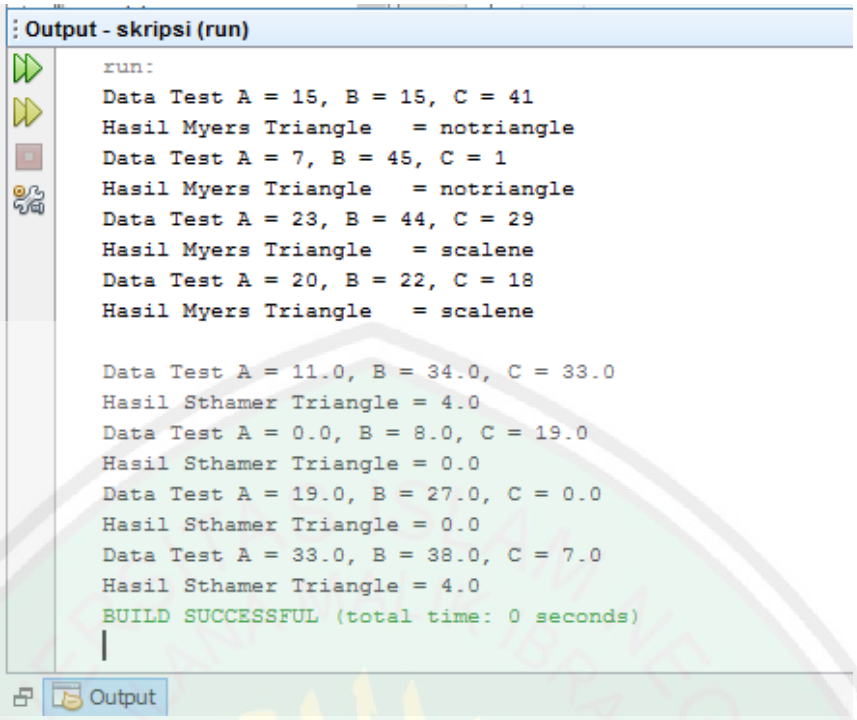
Tabel 4. 2 *Validasi* hasil *Sthamer Triangle*

No	Hasil data *test myers triangle*			*Kualifikasi* hasil
1	A	B	C	
2	11	34	33	Segitiga
3	0	8	19	Bukan segitiga
4	19	27	0	Bukan segitiga
5	33	38	7	Segitiga

Pada Gambar 4. 16, 4. 17, 4. 18 dan 4. 19 menunjukkan hasil data *test sthamer triangle* yang *konvergen* dengan perhitungan metode *Particle Swarm Optimation*. Pada tabel 4. 2 menunjukkan pengujian *kualifikasi* hasil dari dari beberapa hasil *variable* data *test sthamer triangle*. Data *test* A = 33, B = 38 dan C = 7 adalah hasil data *test sthamer triangle* yang terbaik dari beberapa hasil data *test* yang ada, karena hasil data *test* tersebut menunjukkan *kualifikasi* segitiga dengan angka yang paling kecil diantara yang lainnya.

4.3. *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 A, B, dan C terhadap kode program unit (*benchmark*).



Gambar 4. 20 Hasil *validasi* data *test*

Pada gambar 4. 19 merupakan hasil uji *validasi* salah satu data *test* pada kedua *benchmark*. Pada *benchmark Myers Triangle*, uji validasi dilakukan menggunakan salah satu sampel yang berhasil didapatkan pada tabel 4.1, dengan data *test* A=20, B=22, C=18 dimana menghasilkan tipe segitiga tak sama panjang. Pada *benchmark Michael Triangle*, uji *validasi* dilakukan menggunakan salah satu sampel yang berhasil didapatkan pada tabel 4.2, dengan data *test* A=33, B=38, C=7 dimana menghasilkan tipe segitiga.

4.4. Integritas Sains dan Al-Qur'an

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 *test* data yang dibuat *output* secara otomatis. System ini dibuat untuk mempermudah dalam pembuatan teknologi di bidang rekayasa perangkat lunak yang sangat bermanfaat

bagi orang lain. Hal ini dijelaskan dalam kandungan ayat surat Al-Mulk: 19, yaitu :

اَوَلَمْ يَرَوْا اِلَى الطَّيْرِ فَوْقَهُمْ صٰۤفّٰتٍ وَّيَقْبِضْنَ ۛ مَا يُمْسِكُهُنَّ اِلَّا الرَّحْمٰنُ ۗ اِنَّهٗ بِكُلِّ شَيْءٍۢ بَصِيْرٌ

Artinya : "Dan apabila mereka tidak memperhatikan burung-burung yang mengembangkan dan mengatup sayapnya diatas mereka? Tidak ada yang menahan di (udara) selain Yang Maha Pemurah Dia Maha Melihat Segala Sesuatau" (QS. AL-Mulk: 19).

Menurut Tafsir Ibnu Katsir, ayat diatas menjelaskan bahwa terbangnya burung dengan kekuasaan Allah, menunjukkan bahwa dia Maha Melihat setiap perkara yang kecil hingga yang besar. Kemudian Allah berfirman "dan apakah mereka tidak memperhatikan burung-burung yang mengembangkan dan mengatup sayapnya diatas mereka?" yaitu, terkadang burung menegakkan sayapnya di udara dan terkadang melipatnya dan mengembangkanya. "tidak ada yang menahannya," yaitu di udara "selain Yang Maha Pemurah". Karna rahmat serta kelembutannya, dia menundukkan udara untuk burung-burung agar bias terbang menembusnya.

Ayat diatas menjelaskan tentang bagaimana burung bisa terbang mengembangkan sayapnya. Karena burung dilengkapi dengan organ-organ tertentu, misalanya sayap, bulu-bulu yang dapat menahan angin dan badan yang lebih ringan dari pada tenaganya, tentu hal serupa juga tidak mustahil bagi manusia untuk bisa terbang, bila dilengkapi dengan organ-organ yang mampu menerbangkannya. Hal ini pernah dicoba oleh manusia terdahulu ketika mereka mencoba terbang seperti burung. Mereka membuat sayap kemudian dikaitkan

pada kedua tangannya, lalu terbang dari atas, namun sayang mereka tidak bisa terbang ke atas karena tidak seimbang antara berat badan dan kekuatan sayapnya.

Pengembangan ini dilakukan untuk kemajuan teknologi melalui pengujian otomatis pada perangkat lunak menggunakan algoritma *Particle Swarm Optimization* (*PSO).*

BAB V

PENUTUP

5.1 Kesimpulan

Dari hasil penelitian yang telah dilakukan, dapat disimpulkan bahwa pembangkitan data *test* pada pengujian perangkat lunak dengan data *test myers triangle dan sthamer triangle* sebagai data uji atau *inputan*, langkah pertama dengan mengubah data *test* menjadi *Control Flow Graph* (*CFG*) yang kemudian menjadi jalur *path* atau *node* setelah itu diproses menggunaka metode *Particle Swarm Optimation* untuk mencari hasil akhir atau *output* nilai dari variable A, B dan C yang menjadi nilai sisi dari segitiga.

Hasil pengujian *variabel myers triangle* mendapatkan nilai *variable* terkecil dibandingkan *variable* yang lainnya, dan terbukti metode *Particle Swarm Optimation* bekerja dengan menghasilkan *kualifikasi* hasil segitiga tak sama panjang.

Pengujian *variable* pada data *test sthamer triangle* juga sama, menunjukkan hasil yang sesuai dengan jalannya metode *Particle Swarm Optimation*, terbukti juga data *test sthamer triangle* juga menghasilkan data *test* atau *variable* dengan angka yang paling kecil dan *kualifikasi* hasil segitiga.

5.2 Saran

Berikut beberapa saran atau masukan untuk penelitian yang akan dilakukan selanjutnya :

1. *Output* dari aplikasi kurang menjelaskan adanya *iterasi*.

2. Pengujian dapat menggunakan metode lainnya atau dengan mengkombinasi dengan metode yang lain, agar hasil yang didapatkan lebih baik dan lebih lengkap.
3. Pengujian dapat menggunakan jurnal acuan, agar dapat dibandingkan dengan jurnal acuan apakah pengujian yang dilakukan lebih baik dengan jurnal acuan atau tidak.

DAFTAR PUSTAKA

Abdullah, Dr. (2005). *Tafsir Ibnu Katsir Jilid 8*. Tt. Bogor: Pustaka Imam Asy-Syafi'i.

Ahmed, B.S., Zamli, K.Z. (2011). *A variable strength interaction test suites generation strategy using particle swarm optimization*. J. Syst. Softw. 84, 2171–2185.

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.

Bertolino A., Mirandola R., Peciola E. (1997). *A case study in branch testing automation*. J. Syst. Soft. 38(1), 47–59.

Bouchachia, A. (2007). *An immune genetic algorithm for software test data generation*. In: Proceedings of the 7th International Conference on Hybrid Intelligent Systems (HIS'07), pp. 84–89.

Bottaci, L. (2003). Predicate expression cost functions to guide evolutionary search for test data. In *Genetic and Evolutionary Computation Conference*. Springer Berlin Heidelberg, pp. 2455-2464.

Chen, X., Gu, Q., Qi, J., Chen, D. (2010). *Applying particle swarm optimization to pairwise testing*. In: Proceedings of the 34th Annual IEEE Computer Software and Applications Conference (COMPSAC'10), pp. 107–116.

Fabio Palomba , Dario Di Nucci. (2016). *On the Diffusion of Test Smells in Automatically Generated Test Code*: An Empirical Study

Ferrer J., Chicano F., Alba E. (2012). *Evolutionary algorithms for the multi objective test data generation problem*. Softw. Pract. Exp. 42(11), 1331–1362.

G. J. Myers, T. Badgett, T. M. Thomas, and C. Sandler, (2004) *Second Edition The Art of*. John Wiley & Sons, Inc., Hoboken, New Jersey.

Harman M. (2011). *Software engineering meets evolutionary computation*. IEEE Comput. 44(10), 31–39.

Harman M., McMinn P. (2010). *A theoretical and empirical study of search-based testing: local, global, and hybrid search*. IEEE Trans. Softw. Eng. 36(2), 226–247.

Hla, K.H.S., Choi, Y., Park, J.S. (2008). *Applying particle swarm optimization to prioritizing test cases for embedded real time software retesting. In: Proceedings of the 2008 IEEE 8th International Conference on Computer and Information Technology Workshops* (CITWorkshops'08), pp. 527–532.

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 (2014). Pustaka Imam asy-Syafi'i.

Jones B.F., Sthamer H.H., Eyres D.E. (1996). *Automated structural testing using genetic algorithms*. Softw. Eng. J. 11(5), 299–306.

Kaur A., Bhatt D. (2011). *Hybrid particle swarm optimization for regression testing*. Int. J. Comput. Sci. Eng. (IJCSE) 3(5), 1815–1824.

Kennedy, J., Eberhart, R.C. (1995). *Particle swam optimization. In: Proceedings of IEEE International Conference on Neural Networks* (ICNN'95), pp. 1942–1948.

Liang J.J., Qin A.K., Suganthan P.N., Baskar S. (2006). *Comprehensive learning particle swarm optimizer for global optimization of multimodal functions*. IEEE Trans. Evol. Comput. 10(3), 281–295.

Maulana Reza., Romi Satria Wahono., Catur Supriyanto. (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. ISSN 2356-3974

Mao, C., Yu, X., Chen, J. (2012). *Swarm intelligence-based test data generation for structural testing. In: Proceedings of the IEEE/ACIS 11th International Conference on Computer and Information Science* (ACIS-ICIS'12), pp. 623–628.

McMinn, P. (2011). *Search-based software testing: past, present and future. In: Proceedings of ICSE Workshop on the Search-Based Software Testing* (SBST'11), pp. 153–163.

McMinn P. (2004) *Search-based software test data generation: a survey. Softw. Test*. Verif. Reliab. 14, 105–156.

Mandyartha. (2017). *Pembangkitan Data Uji Menggunakan Algoritma Genetika Multi-populasi Fuzzy Adaptif*. 60-69.

Offut, A Jefferson. dan Liu, Shaoying. (1999) Generating test data from SOFL specifications. The Journal of Systems and Software.

Pargas R.P., Harrold M.J., Peck R. (1999). *Automated structural testing using genetic algorithms*. Softw. Test. Verif. Reliab. 9(4), 263–282.

Poli, R. (2008). *Analysis of the publications on the applications of particle swarm optimisation*. J. Artif. Evol. Appl. 2008, 1–10.

R Development Core Team. R. (2010). *A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna.

Pressman. R. S. (2002). *Software Engineering A Practitioner's Approach*.

Pressman. R. S. (2010). *Software Engineering A Practitioner"s Approach Seventh Edition*.

Shi, Y., Eberhart, R.C. (1998). *A modified particle swarm optimizer*. In: Proceedings of the 1998 IEEE International Conference on Evolutionary Computation (ICEC'98), pp. 69–73.

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

Tracey, N., Clark, J., Mander, K., McDermid, J. (1998). *An automated framework for structural test-data generation*. In: Proceedings of the 13th International Conference on Automated Software Engineering (ASE'98), pp. 285–288.

T Manikumar., A John Sanjeev Kumar., R Maruthamuthu. (2016). Automated test data generation for branch testing using incremental genetic algorithm, pp. 959–976

Yuli Agusti, dkk. (2013). *Model Penjadwalan Matakuliah Secara Otomatis Berbasis Algoritma Particle Swarm Optimization*. Vol. 2, No.1.

Y. Singh, *Software Testing*. NewYork. (2012). Cambridge University Press.