Beberapa generator uji data berbasis GA   
Moataz A. *Ahmeda,* *\*,* 1, Irman Hermadib   
*Departemen Informasi dan Ilmu Komputer, King Fahd University of Petroleum and Minerals, Dhahran 31.261, Arab Saudi* b *Departemen Ilmu Komputer, Institut Pertanian Bogor, Bogor 16144, Indonesia* Tersedia online 7 Februari 2007   
**Abstrak** Pengembang telah belajar dari waktu ke waktu bahwa biaya pengujian perangkat lunak sejumlah besar anggaran proyek perangkat lunak. Oleh karena itu, manajer kualitas perangkat lunak telah mencari solusi untuk mengurangi biaya dan waktu pengujian. Mengingat cakupan jalur sebagai kriteria kecukupan uji, kami mengusulkan menggunakan algoritma genetika (GA) untuk mengotomatisasi generasi data uji untuk pengujian kotak putih. Ada bukti bahwa GA sudah berhasil menghasilkan data uji. Namun, generator data uji berbasis GA yang ada mengalami beberapa masalah. Makalah ini menyajikan pendekatan kita untuk mengatasi salah satu masalah ini; Itulah inefisiensi dalam meliput beberapa jalur target. Kami telah merancang generator data uji berbasis GA yang, dalam satu putaran, dapat mensintesis beberapa data uji untuk mencakup beberapa jalur target. Selain itu, kami telah menerapkan serangkaian variasi generator. Hasil percobaan menunjukkan bahwa data uji generator kita lebih efisien dan lebih efektif daripada yang lain.   
2007 Elsevier Ltd. Semua hak dilindungi undang-undang.   
*Kata kunci:* Software pengujian; Pengujian jalan; Algoritma genetika; Uji generator data   
**1. Perkenalan** Pengujian di industri perangkat lunak adalah pekerjaan yang melelahkan dan menyita waktu; mengkonsumsi sekitar 50% dari biaya pengembangan perangkat lunak [1-3]. Secara umum, tujuan dari pengujian perangkat lunak adalah untuk menghasilkan satu set jumlah minimal kasus uji rupa sehingga mengungkapkan sebanyak cacat mungkin dengan memuaskan kriteria tertentu, disebut *kriteria kecukupan tes,* cakupan misalnya jalan.   
Pengujian perangkat lunak otomatis dapat secara signifikan mengurangi biaya pengembangan perangkat lunak.   
Pengujian didefinisikan sebagai proses eksekusi program dengan maksud menemukan kesalahan [3]. Oleh karena itu, sepasang input dan output yang diharapkan, yang disebut *ujian,* dikatakan sukses jika berhasil mengungkap kesalahan, dan bukan sebaliknya. Dengan kata lain, kasus uji yang baik adalah yang memiliki probabilitas tinggi mendeteksi kesalahan yang belum belum ditemukan [3].   
Sebuah datum masukan untuk program diuji, yang merupakan sebuah elemen dari kasus uji, disebut *tes datum.*   
Selama bertahun-tahun, banyak peneliti telah mengajukan berbagai metode untuk menghasilkan data uji / kasus secara otomatis, yaitu berbeda   
metode untuk mengembangkan generator data uji / kasus [4]. Umumnya, mencari sebuah datum masukan dalam domain dari kemungkinan input data ditangani sebagai masalah optimasi [5].   
Para peneliti telah menunjukkan kesesuaian dengan menggunakan perhitungan evolusi dalam pengujian perangkat lunak dan algoritma genetika dikembangkan (GA) berbasis generator data uji [6,7]. Dalam penelitian terbaru, bagaimanapun, Hermadi telah mengamati bahwa GA ada tes berbasis generator data yang dapat menghasilkan hanya satu tes datum pada suatu waktu [8]. Oleh karena itu, dalam usaha untuk menghasilkan satu set data uji (yaitu, lebih dari satu datum uji) untuk memenuhi kriteria tertentu yang dipertimbangkan, generator data uji yang ada harus digunakan lebih dari satu kali (satu untuk setiap datum uji yang diperlukan) . Praktik ini, bagaimanapun, tidak memanfaatkan fakta bahwa beberapa data uji yang diperlukan dapat tersedia sebagai produk sampingan saat mencoba menemukan data uji lainnya. Oleh karena itu, membuat data generator uji yang ada tidak efisien dalam mencoba menghasilkan banyak data uji.

Dalam tulisan ini, kami berfokus pada cakupan jalur sebagai kriteria kecukupan pengujian kami. Tujuan dari penelitian yang dipaparkan di sini adalah untuk tidak melakukan pekerjaan dalam menghasilkan tes dengan menghasilkan beberapa data uji secara efisien untuk cakupan beberapa jalur. Makalah ini menyajikan generator data uji berbasis GA yang mampu menghasilkan, dalam satu GA run, beberapa data uji untuk mencakup beberapa jalur target. Makalah ini juga menyajikan beberapa kandidat fungsi fitness. Kandidat ini mempertimbangkan berbagai blok bangunan, teknik traversal jalan, serta normalisasi, pembobotan, dan skema penghargaan.

Sisa kertas disusun sebagai berikut. Bagian 2 memberikan latar belakang singkat tentang pengujian perangkat lunak.

Bagian 3 membahas pendekatan berbasis GA untuk pembuatan data uji. Ini termasuk diskusi tentang pekerjaan terkait. Bagian 4 menyajikan generator data uji berganda berbasis GA kami beserta variasi yang mungkin digunakan. Bagian 5 membahas berbagai percobaan yang dilakukan untuk secara empiris memvalidasi pendekatan ini. Bagian 6 menyajikan perbandingan dengan pekerjaan lain.

Bagian 7 menyimpulkan makalah ini dan menunjukkan arah yang mungkin untuk penelitian selanjutnya.

2. Pengujian perangkat lunak

Umumnya, teknik pengujian perangkat lunak dikelompokkan menjadi dua kategori: analisis statis dan pengujian dinamis. Dalam analisis statis, pengkodean kode membaca kode sumber program atau perangkat lunak yang diuji (selanjutnya SUT), pernyataan berdasarkan pernyataan, dan secara visual mengikuti aliran logika program dengan memberi masukan. Jenis pengujian ini sangat bergantung pada pengalaman resensi. Metode analisis statis yang umum adalah pemeriksaan kode, panduan kode, dan tinjauan kode [9].

Berbeda dengan analisis statis, yang menggunakan persyaratan program dan dokumen desain untuk tinjauan visual, teknik pengujian dinamis menjalankan program yang diuji pada data masukan uji dan mengamati hasilnya. Biasanya, istilah pengujian mengacu hanya pada pengujian dinamis. Pengujian dinamis dapat dikelompokkan menjadi dua kategori: kotak hitam dan kotak putih. Pengujian kotak putih berkaitan dengan sejauh mana uji coba kasus atau mencakup aliran logika program [9]. Oleh karena itu, jenis pengujian ini juga dikenal sebagai pengujian cakupan logika atau pengujian struktural, karena menganggap struktur program.

Pengujian kotak hitam, alias pengujian fungsional atau berbasis spesifikasi, di sisi lain, menguji fungsionalitas perangkat lunak terlepas dari strukturnya. Pengujian fungsional hanya tertarik untuk memverifikasi output sebagai respons terhadap input data yang diberikan. Makalah ini berfokus pada pengujian cakupan logika. Kecukupan pengujian cakupan logika dapat dinilai dengan menggunakan kriteria yang berbeda: pernyataan, keputusan (alias cabang), kondisi, keputusan / kondisi, cakupan multi-kondisi, dan cakupan jalur [9,10].

Kriteria cakupan pernyataan mengharuskan setiap pernyataan dalam program dijalankan setidaknya satu kali. Kriteria yang lebih kuat dikenal sebagai keputusan atau cakupan cabang. Kriteria ini mengharuskan setiap keputusan memiliki hasil yang BENAR dan SALAH setidaknya satu kali. Kriteria yang terkadang lebih kuat dari pada cakupan keputusan adalah cakupan kondisi. Kriteria ini mengharuskan setiap kondisi dalam sebuah keputusan mengambil semua kemungkinan hasil setidaknya satu kali. Meskipun kriteria cakupan kondisi tampaknya memenuhi kriteria cakupan keputusan, namun tidak selalu dilakukan. Jika keputusan IF (A AND B) sedang diuji, kriteria cakupan kondisi akan memungkinkan seseorang untuk menulis dua kasus uji - A BENAR, B adalah FALSE, dan A adalah FALSE, B BENAR - tapi ini tidak akan menyebabkan THEN Klausa pernyataan IF untuk dieksekusi. Cara yang jelas dari dilema ini adalah kriteria yang disebut cakupan keputusan / kondisi. Ini memerlukan kasus uji yang memadai sehingga setiap kondisi dalam sebuah keputusan mengambil semua hasil yang mungkin paling tidak satu kali, setiap keputusan mengambil semua hasil yang mungkin setidaknya satu kali, dan setiap titik masuk dipanggil setidaknya satu kali.

Kelemahan dengan cakupan keputusan / kondisi adalah bahwa meskipun tampaknya menampilkan semua hasil dari semua kondisi, seringkali tidak karena kondisi tertentu menutupi kondisi lain. Selain itu, kesalahan dalam ekspresi logis tidak harus dibuat terlihat oleh cakupan kondisi dan kriteria cakupan keputusan / kondisi [9]. Kriteria yang mengatasi masalah ini adalah cakupan multi-kondisi. Kriteria ini mengharuskan seseorang untuk menulis kasus uji yang memadai sehingga semua kombinasi hasil kondisi yang mungkin terjadi di setiap keputusan, dan semua titik masuk, dipanggil setidaknya satu kali. Cakupan yang lebih praktis, banyak digunakan dan efektif dicapai dengan cakupan jalur [10,11]. Kriteria kelayakan uji cakupan jalan berkaitan dengan pelaksanaan jalur (terpilih) dalam program. Dalam tulisan ini kami menerapkan kriteria cakupan jalur sebagai tujuan pengujian kami. Karena dalam sebuah program dengan loop, eksekusi setiap jalur biasanya tidak memungkinkan, pengujian jalur yang lengkap tidak dipertimbangkan dalam kasus seperti layak untuk diuji, yaitu tidak dapat dicapai dalam waktu yang dapat diterima dengan sumber daya yang ada.   
**3. Uji data generator** Berbagai metode untuk mengembangkan tes generator data / kasus telah diajukan selama bertahun-tahun [12,13]. Setiap metode dimaksudkan untuk memenuhi kriteria uji kecukupan tertentu, dan sesuai dengan tujuan pengujian yang diinginkan [10,12,13].   
Pada usia dini otomatisasi pengujian perangkat lunak, sebagian besar generator data uji yang menggunakan algoritma gradient descent [12,14,15]. Namun, algoritma ini tidak efisien, dan memakan waktu, dan tidak bisa melarikan diri dari optima lokal dalam ruang pencarian mungkin input data [7,12,16]. Dengan demikian, algoritma pencarian meta-heuristik telah digunakan dalam generator data uji sebagai alternatif yang lebih baik [7,13]. Wegener dkk. telah menunjukkan kesesuaian dengan menggunakan algoritma evolusioner dalam pengujian perangkat lunak [7].   
Proses pembuatan data uji otomatis menggunakan GA memiliki dua langkah utama:   
1. *Instrumentasi:* Instrumentasi adalah, pada dasarnya, proses memasukkan probe (tag) pada awal / akhir dari setiap blok kode, misalnya pada awal setiap fungsi, sebelum dan sesudah hasil benar dan salah dari setiap kondisi.   
Sebagai contoh, dengan menggunakan kriteria uji cakupan jalan, tag ini digunakan untuk memantau jalur yang dilalui dalam program saat dijalankan dengan data masukan uji tertentu.   
2. *Uji evolusi Data:* Ini adalah lingkaran di mana program ini dijalankan dengan beberapa input data awal (acak atau unggulan), umpan balik dikumpulkan, dan input data disesuaikan sampai kriteria yang memuaskan tercapai. Umpan balik adalah semacam nilai fitness yang diberikan pada data masukan saat ini untuk mencerminkan kesesuaiannya sesuai dengan kriteria pengujian yang diberikan.   
GA didasarkan pada teori evolusi [17]. Langkah-langkah dasar dari GA adalah berikut [13,16]:   
1. Buat populasi awal dari solusi kandidat.   
2. Hitung nilai fitness masing-masing kandidat tersebut.   
3. Pilih semua kandidat yang memiliki nilai fitness di atas atau di ambang batas.   
4. Buat perturbasi pada masing-masing kandidat terpilih menggunakan operator genetika, misalnya crossover dan mutasi.   
Langkah-langkah ini, kecuali tahap inisialisasi pertama, diulangi sampai solusi kandidat menjadi solusi.   
Sebelum GA dapat digunakan, ada empat hal yang bergantung pada domain: mendefinisikan representasi genetik dari solusi masalah, menentukan fungsi fitness, memilih metode seleksi kandidat, dan menentukan operator genetik.   
Generator data uji berbasis GA menggunakan algoritma genetika sebagai mesin pencari utamanya dalam mencari semua data uji yang sesuai sesuai dengan kriteria kecukupan uji. Periset telah melakukan beberapa pekerjaan dalam mengembangkan generator data uji berbasis GA. Dalam sekuel ini, kami menyajikan ringkasan diskusi tentang beberapa karya yang ada yang kami anggap penting sehubungan dengan subjek yang sedang dibahas.   
Pei dkk. [18] pada tahun 1994 mengamati bahwa sebagian besar generator data uji, yang dikembangkan di era mereka, menggunakan evaluasi simbolis. Mereka mengamati bahwa pengujian dinamis berbasis statis dan gradien-turunan dikembangkan.   
Namun, mereka menyimpulkan bahwa pengujian statis tidak praktis, sedangkan yang dinamis tidak efektif. Kekurangan ini telah mengilhami Pei dkk. Untuk mengembangkan generator data uji jalur tunggal yang menggunakan GA.   
Sekitar waktu yang sama, Roper dkk. [2] pada tahun 1995 mengembangkan tes berbasis GA Generator data yang memiliki tujuan untuk melintasi semua cabang dalam program target. Generator mereka mengambil sebuah program dan instrumen secara otomatis dengan probe untuk memberikan umpan balik mengenai cakupan cabang yang diraih.   
Setahun kemudian, Jones dkk. [5] pada tahun 1996 mengembangkan sejenis berbasis GA Generator data uji untuk mencapai cakupan cabang.   
Kontribusi utama mereka adalah penggunaan sekuens string biner untuk representasi individual, yang dikonversi ke bilangan desimal sebelum eksekusi program, dan penggunaan grafik aliran kontrol yang tidak dikontrol (unrolled control flow graph / CFG) untuk mewakili satu, dua, atau lebih iterasi untuk Setiap loop, yang membuat CFG asyclic. Karena masing-masing cabang dijalankan, generator data uji secara otomatis melintasi CFG ke cabang berikutnya dengan cara yang lebih luas.   
Michael et al. [19-24] pada tahun 1997, 1998, dan 2001 menerapkan pendekatan fungsi minimisasi Korel ini [15] di-berdasarkan GA pembangkit uji data mereka. Mereka telah membangun sebuah generator data uji yang disebut GADGET (Genetic Algorithm Data Generation Tool), yang memiliki kemampuan untuk instrumen program secara otomatis tanpa batasan dalam bahasa pemrograman, namun memiliki batasan bahwa ia hanya dapat menerima input skalar. GADGET memiliki cakupan keputusan kondisi sebagai kriteria kecukupannya. GADGET menggunakan GA sederhana dan juga GA diferensial. Perbedaan antara diferensial GA dan GA sederhana adalah dalam proses rekombinasi [20]. Hasil Michael et al menunjukkan bahwa, pada umumnya, GA sederhana mengungguli diferensial satu. GADGET dianggap sebagai generator data uji pertama yang diuji terhadap program dunia nyata bernama b737, yang merupakan bagian dari sistem autopilot (perangkat lunak kontrol dunia nyata). Michael et al. Melaporkan bahwa kinerja generasi uji acak memburuk untuk program yang lebih besar. Pekerjaan yang dilakukan oleh Pargas dkk. [25] pada tahun 1999 merupakan perbaikan terhadap karya Jones et al. Pendekatan yang mereka berikan juga menggunakan informasi cabang untuk mengevaluasi fungsi fitness, kecuali menggunakan grafik ketergantungan kontrol untuk evaluasi fitness, yang mereka klaim dapat memberikan evaluasi fitness yang lebih tepat daripada pendekatan Jones et al. Dan Michael et al. . Pargas et al. GA yang diparalelkan untuk membuatnya lebih cepat dan juga mengklaim bahwa pendekatan tersebut dapat memberikan cakupan jalur dengan sedikit modifikasi.

Bueno dan Jino [26] pada tahun 2000 mengusulkan sebuah pendekatan yang memanfaatkan informasi arus informasi kontrol dan arus. Pendekatan yang diusulkan dimaksudkan untuk memenuhi pengujian cakupan jalan. Selain itu, juga menangani identifikasi jalur program yang berpotensi tidak dapat dicapai dengan memantau kemajuan pencarian data uji yang diperlukan. Pendekatan ini mempertimbangkan peningkatan fitness pasien secara terus-menerus sebagai indikasi bahwa jalan yang layak ditutupi. Di sisi lain, upaya untuk menghasilkan data uji untuk hasil jalur yang tidak tepat, selalu, dalam kemajuan yang terus-menerus. Lin dan Yeh [11] pada tahun 2000 memperluas pekerjaan yang dilakukan oleh Jones et al. Dalam pekerjaan mereka, mereka menggunakan kriteria cakupan jalur daripada cakupan cabang. Mereka juga memperpanjang jarak hamming biasa (tertimbang) sehingga dapat menangani pemesanan yang berbeda dari jalur target yang memiliki simpul cabang yang sama. Alasannya di sini adalah bahwa, dalam pengujian jalur, dua jalur yang berbeda mungkin berisi cabang yang sama namun dalam urutan yang berbeda, di mana jarak hamming sederhana tidak lagi sesuai. Mereka menamai fungsi fitness SIMILARITAS, karena menghitung item yang serupa sehubungan dengan pemesanan mereka di dalam dua jalur yang berbeda, yaitu cabang, antara jalur yang dijalankan saat ini dan jalur target. SIMILARITAS yang lebih besar menyebabkan fitness lebih baik.

Wegener dkk. [27] pada tahun 2002 mengembangkan generator data uji GA berbasis otomatis untuk pengujian perangkat lunak struktural, khususnya cakupan pernyataan dan cabang, sistem perangkat lunak tertanam di dunia nyata. Fungsi fitness mereka terdiri dari dua blok bangunan utama: tingkat aproksimasi dan predikat jarak lokal yang dinormalisasi. Tingkat aproksimasi menunjukkan jumlah node percabangan yang terus menerus antara cabang yang dilalui oleh individu dan cabang target (mereka menyebutnya "tujuan parsial"). Jarak lokal dihitung untuk individu dengan cara kondisi percabangan pada nodus percabangan di mana simpul target tidak terjawab. Sayangnya, laporan tersebut tidak menggambarkan skema normalisasi dari nilai jarak lokal. Nilai fitness keseluruhan adalah penjumlahan nilai tingkat aproksimasi dan nilai jarak lokal. Meskipun alat mereka bekerja hanya pada satu tujuan parsial setelah yang lain, mempertimbangkan pelaksanaan uji datum yang biasanya mengarah pada beberapa tujuan parsial. Dengan demikian, tes segera berfokus pada tujuan parsial yang sulit dijangkau. Kriteria penghentian yang digunakan adalah pernyataan lengkap / cakupan cabang dan jumlah generasi; Mana yang puas dulu

Kertas Wegener el al tidak membahas apakah beberapa target dapat ditutupi dalam satu putaran. Namun, pendekatannya, atau lebih tepatnya "kontrol uji", mengevaluasi semua individu yang dihasilkan sehubungan dengan semua target yang tidak dapat diraih. Dengan demikian, target lain yang ditemukan secara kebetulan diidentifikasi, dan individu dengan nilai fitness yang baik untuk satu atau lebih target dicatat dan disimpan untuk melakukan pembekuan pengujian berikutnya berikutnya terhadap target yang tidak ditemukan. Selanjutnya, mereka melaporkan bahwa cakupan penuh beberapa program tercapai, namun tidak untuk semua program sekalipun. Menurut makalah mereka, mereka menyelidiki apakah pernyataan / cabang yang tidak layak atau jumlah generasi adalah beberapa alasan mengapa tidak dapat mencapai cakupan penuh dalam beberapa program.

Hermadi dan Ahmed dalam makalah mereka yang diterbitkan pada tahun 2003 [28] mengajukan sebuah pendekatan yang memenuhi kriteria cakupan jalur. Pekerjaan mencoba menghasilkan beberapa data uji untuk mencakup beberapa jalur target sekaligus. Makalah tersebut melaporkan hasil penelitian awal yang mengarah pada karya yang disajikan dalam makalah ini. Diskusi yang lebih rinci dan evaluasi menyeluruh terhadap generator data uji berbasis GA yang ada terdapat di Hermadi [8]. Hermadi telah menunjukkan beberapa kekurangan terkait pendekatan yang ada. Memuaskan beberapa jalur target sekaligus merupakan salah satu kekurangannya. Kami menyajikan usaha kami untuk mengembangkan beberapa generator kasus uji berbasis GA di bagian berikutnya.

4. Beberapa jalur pengujian

Seperti telah ditunjukkan, banyak generator data uji berbasis GA mengadopsi pernyataan atau cakupan cabang sebagai tujuan mereka. Namun, secara umum, kriteria cakupan jalan meliputi kriteria cakupan pernyataan dan cakupan cabang; dengan demikian, pengujian struktural software lebih efektif harus memiliki cakupan jalan sebagai tujuan [29,30] Sebagai bagian sebelumnya menyimpulkan, tidak ada karya-karya yang cakupan jalur memuaskan berfokus beberapa target yang memuaskan   
Path pada satu waktu, yaitu meliputi satu set jalur target dalam satu run GA. Jelas, mencakup banyak jalur dalam satu putaran akan memerlukan penggabungan jalur ini dalam perhitungan fitness. Alasan di balik mempertimbangkan beberapa jalur dalam satu proses didasarkan pada pengamatan bahwa ketika mencoba untuk mencakup satu jalur tunggal, beberapa individu yang dihasilkan mencakup jalan lain sebagai produk sampingan. Dengan demikian, mencoba untuk memenuhi banyak jalur sekaligus diharapkan dapat meningkatkan efektivitas dan efisiensi generator data uji, yaitu mencapai cakupan lebih banyak dengan sumber daya yang lebih sedikit daripada   
Generator uji jalur tunggal yang membutuhkan banyak jalan untuk mencakup jumlah jalur yang sama. Pada Bagian 4.1, kami membahas berbagai keputusan yang kami pertimbangkan dalam merancang fungsi fitness untuk generator data uji berbasis GA kami. Kami juga menyajikan serangkaian variasi fungsi fitness yang berbeda.   
*4.1.* *Desain fungsi fitness* Pada sebagian besar teknik pencarian meta-heuristik, terutama GA, tujuan pengujian dimodelkan sebagai fungsi fitness yang harus dioptimalkan untuk menemukan data uji yang diinginkan. Cara di mana heuristik dari masalah pembuatan data uji dimasukkan ke dalam fungsi fitness kontribusi signifikan terhadap kinerja generator data uji [31-33].   
Hermadi telah mengidentifikasi beberapa atribut penting dari fungsi fitness yang dapat digunakan dalam membimbing desain fungsi fitness yang baik dalam hal efektivitas dan efisiensi pencarian: blok bangunan, normalisasi, metode traversal jalur, pengaruh lingkungan, penyeimbang / pembobotan, penyesuaian, dan menguntungkan [8]. Atribut ini memandu keputusan desain yang akan dibuat saat mengembangkan generator data uji berbasis GA untuk cakupan multi-jalur.   
*Bangunan blok:* Sebuah blok bangunan merupakan konstituen dari fungsi fitness. Konstituen fungsi fitness mempengaruhi keefektifan / efisiensinya dalam mengarahkan pencarian ke arah tujuan yang diinginkan. Blok bangunan dasar dari calon fungsi fitness yang diusulkan kami didasarkan pada membandingkan jalur yang dilalui (yaitu, jalur yang ditawarkan oleh individu GA dalam populasi) untuk menargetkan jalur dari segi jarak *D* dan pelanggaran *V.* *D* dihitung sebagai perbedaan antara jalur yang dilalui dan jalan target, dalam hal nilai-nilai predikat untuk “tak tertandingi simpul-cabang”. Sebagai contoh, perhatikan predikat “A *<B”,* dan menganggap bahwa predikat ini harus “false” dalam jalan target. *D* akan dihitung sebagai ABS *(B* - *A)* jika A adalah benar-benar kurang dari B di jalur yang dilalui diberikan. *V* memberitahu berapa banyak node yang tak tertandingi ada antara jalan target dan yang dilalui. Perlu dicatat di sini bahwa *D* 0 dan *V* 0. Sebuah node adalah predikat bercabang, yaitu   
Sebuah pernyataan dimana program tersebut menuju ke cabang yang berbeda secara logis. Misalnya, pernyataan IF-THEN-ELSE adalah simpul yang memiliki dua cabang logis, yaitu cabang THEN dan cabang ELSE. Tujuannya adalah untuk meminimalkan jarak *D* dan pelanggaran *V.* Pers. (1) menunjukkan blok bangunan dari fungsi fitness yang diusulkan:   
*JIKA* *ij* = *Dij* + *Vij,* (1)   
*dij* =   
*N* *k* = 1   
*Dij.k,* (2)   
*Vij* =   
*N* *k* = 1   
*Vij.k,* (3)   
di mana *i* adalah indeks dari jalan target; *j* adalah indeks individu; *k* adalah indeks simpul di kedua jalan target *saya* dan dilalui jalur *j;* dan *JIKA* adalah fitness menengah yang terlihat pada fitness dari individu terhadap satu jalan target. *JIKA* dianggap sebagai sebuah blok bangunan untuk fitness individu secara keseluruhan di mana fitness sehubungan dengan semua jalur target dipertimbangkan.   
Kami menghitung nilai predikat menggunakan fungsi jarak Korel ini [15]. Jarak sama dengan nol jika simpul-cabang dari kedua jalur target dan jalur yang dilalui cocok.   
Sebagai contoh, perhatikan Gambar. 1, di mana B1, B2, dan B3 menunjukkan nomor cabang, T dan F berdiri untuk Benar dan Salah, masing-masing, dan nomor yang menempel pada ujungnya nomor tepi, yaitu untuk menggambarkan jalur yang diambil oleh datum masukan.   
Mari kita asumsikan: B1 adalah “indexlength”, B2 adalah “max *<nomor* *(indeks)”,* dan B3 adalah *“min>* nomor *(indeks)”.* Mempertimbangkan situasi yang diberikan oleh individu tertentu (katakan individu nomor 2) sebagai: index = 1, panjang = 2, max = 10, min = 5, nomor *(1)* = 2. sesuai dilalui jalur *(TR* *2)* dalam hal ini akan {1 T 2 F 3 T} (yaitu, berlaku untuk Cabang 1, palsu untuk Cabang 2, dan berlaku untuk Branch 3). Pertimbangkan sasaran jalan ini: *TG* 1 = {1 T 2 F 3 T}. Menurut Persamaan. (1) - (3), jarak antara *TG* 1 dan *TR* 2, yaitu *D* 12, sama dengan penjumlahan dari semua jarak, yaitu jarak B1 *(DB* *1),* jarak B2 *(DB* *2),* dan jarak B3 *(DB* *3).* Menurut fungsi jarak Korel ini, jarak akan *DB* 1 = 0, *DB* 2 = 0, dan *DB* 3 = 0; dan karenanya *D* 12 = *DB* 1 + *DB* 2 + *DB* 3 = 0.In casewe memiliki individu lain, mengatakan *TR* 3, yang memiliki pasangan variablevalue sama dengan *TR* 2 kecuali jumlah *(1)* = 6; dengan demikian, *D* 13 = *DB* 1 + *DB* 2 + *DB* 3 = 0 + 0 + ABS *(min* - nomor *(1))* = 1, dan *V* 13 = 1, karena hanya B3 telah dilanggar.   
*Normalisasi:* *D* dan *V* dapat dihitung sebagai baik nilai polos atau normal. Ukuran jarak polos (yaitu, mutlak) sama dengan penjumlahan semua nilai predikat cabang-cabang simpul yang tak tertandingi. Di sisi lain, pelanggaran biasa sama dengan jumlah simpul tak tertandingi yang ditentukan oleh pendekatan jalur traversal, yaitu path-wise atau predicate-wise seperti yang dibahas di bawah ini. Jelas, kedua jarak polos dan nilai pelanggaran tidak dibatasi. Nilai blok bangunan dapat dinormalisasi di seluruh individu populasi. Normalisasi akan memungkinkan perbandingan yang lebih bermakna antara fungsi fitness individu yang berbeda. Kami melakukan perbandingan antara fungsi fitness yang menggunakan blok bangunan biasa dibandingkan dengan yang menggunakan normalisasi. Dengan normalisasi, jarak *D* dinormalkan dengan baik semua jalur target atau oleh semua jalur sasaran serta individu lainnya. Pelanggaran *V,* di sisi lain, adalah dinormalisasi oleh panjang, yaitu jumlah cabang, jalan target.   
*Metode traversal:* Metode traversal adalah cara mengukur “kedekatan” antara jalan yang dilakukan oleh individu, yaitu input data yang dihasilkan, dan jalan target. Ada dua cara untuk menghitung "kedekatan" semacam itu: bijak dan bijaksana (juga pernyataan bijaksana dan predikat bijaksana). Dengan menggunakan metode traversal path-wise, fungsi fitness tidak mempertimbangkan jalur yang sesuai (sub) setelah penyimpangan pertama (yaitu simpul yang tak tertandingi). Sementara, dengan menggunakan metode predikat-bijaksana, fungsi fitness mempertimbangkan jalur yang sesuai (sub) setelah sejumlah penyimpangan.   
*Lingkungan pengaruh:* Dalam mencoba untuk menghasilkan beberapa data tes dalam satu run, fungsi fitness harus mempertimbangkan tekanan persaingan antara individu-individu untuk menutupi target. Ada dua jenis nilai fitness dari seorang individu: nilai fitnessnya sendiri dan nilai fitnessnya yang dipengaruhi oleh target dan / atau individu lain yang bersaing untuk menutupi target yang sama. Dengan demikian, kita mempertimbangkan dua skema normalisasi jarak: didasarkan pada jalur sasaran (kami menyebutnya sebagai *Op)* saja; dan berdasarkan kedua jalur target dan individu lain (kita menyebutnya sebagai *Oc).* Dalam *Op,* kita menormalkan jarak lebih dari saat ini semua jalur sasaran ditemukan, sementara *Oc* menormalkan jarak lebih baik saat semua jalur sasaran ditemukan dan semua kromosom bersaing yang ada dalam populasi.   
*Pembobotan:* Heuristik mungkin mengatakan bahwa blok bangunan yang berbeda dari fungsi fitness harus memiliki bobot yang berbeda berkenaan dengan kontribusi nilai keseluruhan fitness. Kami menggunakan bobot untuk memungkinkan diferensiasi antara kontribusi dari jarak *D* dan pelanggaran *V* untuk fitness secara keseluruhan.   
*Penyesuaian:* Ini berlaku untuk setiap blok bangunan dari fungsi fitness dan / atau fungsi fitness secara keseluruhan, sesuai dengan kriteria yang dipilih didefinisikan oleh perancang tes pembangkit. Operasi penyesuaian dapat berupa penambahan, perkalian, atau tindakan apa pun yang diperlukan (misalnya perkalian dari setiap blok bangunan dengan nomor yang dipilih) perancang mempertimbangkan diperlukan untuk menyempurnakan fungsi fitness.   
*Menghargai:* Untuk setiap jalan target, individu yang memiliki fitness tertinggi sehubungan dengan yang jalan target dibandingkan dengan individu lain dalam populasi adalah pemenang berkaitan dengan yang jalan target, yaitu itu adalah salah satu yang keseluruhan fitness mendapat hadiah; Itu adalah keseluruhan fitnessnya akan terpengaruh secara positif. Alasannya, di sini, adalah memberi kesempatan yang lebih baik kepada individu tersebut untuk bertahan hidup ke generasi berikutnya karena ini adalah jalur paling dekat dengan beberapa jalur target.   
*Perhitungan fitness akhir:* Dalam rangka untuk mendapatkan nilai keseluruhan fitness kita harus meringkas dan menormalkan fitness seluruh menengah, yaitu *IF,* nilai untuk setiap kromosom.

*4.2.* *Kemungkinan fungsi fitness* Berdasarkan pembahasan di atas pada keputusan yang berbeda yang harus dilakukan berkenaan dengan functiondesign fitness, Tabel 1 daftar variasi mungkin menunjukkan bersama dengan nilai-nilai yang sesuai mereka. Demi referensi yang lebih mudah, singkatan dari nilai yang mungkin ditunjukkan di bawah kolom "Kode" di tabel. Tabel tersebut menunjukkan bahwa ada 128 *(=* 2 \* 2 \* 2 \* 4 \* 2 \* *2)* kemungkinan fungsi fitness untuk semua kombinasi   
Atribut ini   
Berdasarkan intuisi kami, kami berharap bahwa nilai yang dinormalisasi akan lebih bermakna daripada yang biasa. Dengan demikian, untuk mengurangi jumlah kandidat fungsi fitness yang mungkin untuk diselidiki, kami membatasi ruang lingkup properti-properti di mana nilai yang mungkin dinormalisasi. Dengan cara ini, kita akhirnya memiliki 32 *(=* 2 \* 2 \* 4 \* *2)* calon fungsi mungkin fitness.   
Rincian lebih lanjut dan diskusi formal mengenai rumus matematika yang mengatur fungsi fitness komposisi dan perhitungan dapat ditemukan di Hermadi [8].   
**5. Percobaan** Pada bagian ini, kami menyajikan dan menilai kinerja (yaitu, kekuatan dan kelemahan) dari semua fungsi fitness yang diusulkan (misalnya, kandidat) dengan menggunakan beberapa SUT. Bagian ini juga membahas penerapan generator data uji berbasis GA kami, termasuk masalah disain, penyiapan, dan implementasinya. Akhirnya, kami menyajikan hasil dan analisis eksperimental.   
*5.1.* *Desain eksperimen* Kami telah melakukan tujuh eksperimen berbeda dengan menggunakan Matlab; Setiap percobaan mempertimbangkan satu SUT. Setiap eksperimen terdiri dari serangkaian rangkaian; Satu set untuk setiap fungsi fitness. Setiap rangkaian terdiri dari setidaknya 10 putaran, di mana kinerja rata-rata di setiap set dilaporkan untuk setiap fungsi fitness. Fungsi fitness 32 calon dibedakan dari satu sama lain dengan pengaturan atribut mereka (yaitu, nilai-nilai yang ditetapkan untuk poin variasi yang dibahas di bagian sebelumnya) seperti yang ditunjukkan pada Tabel 2.   
Kami terutama menilai kinerja melalui tiga grafik: generasi-ke-generasi (G2G) prestasi, fitness terbaik, dan konvergensi klaster *(phi).* Prestasi G2G digunakan untuk menganalisis efektivitas dan efisiensi setiap fungsi calon fitness, sementara grafik klaster konvergensi digunakan untuk menganalisis eksplorasi dan eksploitasi perilaku masing-masing calon fungsi fitness. Grafik fitness terbaik dimaksudkan untuk menganalisis perilaku solusi kandidat terbaik dari generasi ke generasi. Memiliki grafik ini membantu dalam membandingkan kandidat yang berbeda. Rincian lebih lanjut tentang jenis grafik dapat ditemukan di [30]. Kami telah merencanakan grafik yang sesuai untuk setiap eksperimen. Namun, karena terbatasnya jumlah halaman kertas, kami hanya menampilkan dua set grafik untuk dua SUT dari tujuh SUT. Pembaca disarankan untuk berkonsultasi Hermadi [8] untuk set lengkap grafik.

*5.2.* *Persiapan SUT* Kami telah memilih tujuh program uji sebagai SUT untuk eksperimen; ini adalah: *tc,* *bs,* *adalah,* *ns,* *mm* - *t,* *mm* - *f,* dan *mm* - *i.* Program-program dan target mereka jalan yang sesuai adalah sama dianggap oleh peneliti lain [2,5,11,18,24-28].   
1. *Minimum* - *maksimum* *(mm):* Mengingat array angka, *mm* adalah sebuah program untuk menemukan theminimum nomor andmaximum dalam array. Program ini memiliki dua pernyataan pilihan berurutan di dalam satu lingkaran dimana semua kondisi (predikat) sederhana / primitif. Selama percobaan kami, kami membiarkan panjang array menjadi variabel, dan membatasi konten array ke bilangan bulat. *mm-f* dan *mm-i* adalah variasi dari *mm.* *mm-f* adalah *mm* dengan semua jalur sasaran adalah yang layak, sementara *mm-i* adalah *mm* dengan beberapa jalur sasaran adalah yang tidak layak.   
2. *Segitiga classifier* *(tc):* Mengingat tiga angka, *tc* adalah program untuk mengklasifikasikan apakah angka-angka ini membentuk segitiga atau tidak. Jika ya, maka program menentukan apakah segitiga itu scalene, isosceles, atau equilateral. Segitiga classifier memiliki tiga pernyataan seleksi bersarang di mana semua keputusan adalah predikat senyawa.

3. *Bubble sort* ( *bs* ): Mengingat array angka, *bs* adalah program untuk mengurutkan angka-angka ini dalam urutan meningkat. Program ini memiliki dua loop yang bersarang dan satu seleksi yang bersarang di dalam loop batin. Loop luar mengandung predikat majemuk.   
4. *Penyisipan semacam* ( *adalah* ): Mengingat array angka, *adalah* merupakan program untuk mengurutkan angka-angka ini dalam urutan meningkat. Program ini memiliki dua loop yang bersarang dan satu seleksi yang bersarang di dalam loop batin. Loop batin mengandung predikat majemuk.   
5. *Pencarian biner* ( *ns* ): Mengingat array angka dan kunci, *ns* adalah sebuah program untuk menemukan kunci di antara angka-angka ini. Program ini memiliki loop tunggal yang berisi pilihan tunggal.   
6. *Minimum* - *maksimum dan segitiga classifier* ( *mm-t* ): Mengingat tiga angka, program ini dikombinasikan output kedua minimum-maksimum dan klasifikasi segitiga juga. Program ini terbentuk dari *mm* dan *tc* untuk memungkinkan lebih banyak kompleksitas ketika menyelidiki kinerja fungsi calon fitness kami.   
Setiap SUTs atas pose karakteristik khusus yang kami ingin menyelidiki kinerja fungsi calon fitness kami melawan. Untuk setiap SUT, kita instrumen manual program asli tanpa mengubah nya semantik. Kemudian, kita membangun sesuai CFG dan menghasilkan daftar path target yang dipilih dari itu. Kami mengembangkan pembangkit uji data kami menggunakan Matlab. Kami juga mengembangkan versi diinstrumentasi dari SUTs dipertimbangkan dalam Matlab

*5.3.* *GA parameter konfigurasi*Dalam menggunakan GA, nilai-nilai parameter GA harus diatur sebelum tangan. Tabel 3 menunjukkan nilai-nilai untuk semua parameter GA yang kita gunakan. Pemilihan nilai-nilai ini tunduk pada praktek trial-and-error.   
Parameter ini, bagaimanapun, sedikit berbeda dari satu percobaan ke yang lain berdasarkan yang sesuai SUT. Dengan kata lain, pengaturan yang sama tidak (tidak bisa) diterapkan di semua SUTs. Keuntungan dari variasi ini adalah bahwa memberikan gambaran tentang bagaimana untuk mengatur parameter ketika berhadapan dengan program tes lain yang memiliki karakteristik serupa dengan SUTs kami telah digunakan untuk eksperimentasi.   
Perlu dicatat di sini bahwa pengaturan parameter tidak hanya tergantung pada karakteristik SUT, tetapi juga pada calon fungsi fitness diadopsi serta jumlah jalur sasaran yang dipertimbangkan. Misalnya, SUT lebih rumit dengan jumlah yang lebih besar dari jalur target diperkirakan membutuhkan ukuran populasi yang lebih besar dan jumlah yang lebih besar dari generasi untuk menemukan data uji efektif (lihat Gambar. 2 , di mana PF dan LG merupakan jumlah jalur ditemukan dan generasi terakhir di mana jalan target layak terakhir ditemukan, masing-masing). Ara. 2 menunjukkan bahwa *ns* adalah pengecualian dalam arti tidak mendukung harapan bahwa semakin banyak jalur, semakin banyak jumlah generasi. Hal ini karena jalur   
dari *ns* dianggap lebih mudah daripada SUTs lainnya.  
*5.4.* *GA dan fitness parameter fungsi pengaturan*Berdasarkan hasil awal trial-and-error, kami memilih tingkat menjadi: baik 0,1 atau 0,3 untuk berat statis, baik 0,5 atau 0,9 crossover, dan baik 0,1 atau 0,3 untuk mutasi. Alasan di balik pemilihan berat statis adalah bahwa jarak predikat kontribusi jauh lebih sedikit daripada pelanggaran. Adapun crossover dan mutasi tingkat, kami mencoba untuk menjaga keseimbangan antara eksplorasi dan eksploitasi dari ruang pencarian.   
Sebagai pra-percobaan untuk menemukan kombinasi terbaik untuk menggunakan, kita menerapkan semua parameter nilai kombinasi untuk semua calon fungsi 32 fitness menggunakan minimum-maksimum ( *mm-i* ) sebagai SUT dengan jalan tidak layak termasuk dalam set path sasaran. Kami menggunakan jumlah keberhasilan, yaitu jumlah calon fungsi fitness yang ditemukan semua jalur sasaran layak yang diperlukan, untuk menilai efektivitas setiap kombinasi parameter-nilai.   
Dua kombinasi ini memiliki jumlah yang sama dari keberhasilan: 0.1-0.5-0.3 dan 0.1-0.9-0.3. Dengan demikian, kita hanya sewenang-wenang memilih satu dari mereka, yaitu 0.1-0.9-0.3. Alasan di balik seleksi ini adalah bahwa hal itu akan memungkinkan lebih banyak eksplorasi karena tingkat Crossover yang lebih tinggi.   
Tabel 4 daftar semua percobaan yang dilakukan, di mana setiap percobaan terdiri dari 10 berjalan per fungsi fitness calon; kecuali untuk nos. 4, 5, dan 7; yang terdiri dari 20 berjalan masing-masing untuk memungkinkan lebih percaya diri dalam rata-rata yang dilaporkan.   
Percobaan 4 dan 5 dimaksudkan untuk melihat pengaruh dari jalur tidak layak pada perilaku calon fungsi fitness yang berbeda.   
Jalur yang sudah dibahas akan disalin ke daftar jalur sukses bersama dengan data pengujian yang sesuai mereka; jalur ini dihapus dari daftar jalur sasaran yang belum tertutup. Oleh karena itu, selama tes pembuatan data, terdapat dua daftar: jalur target tertutup dan (belum-to-be-tertutup) sasaran jalur. Pada akhirnya, jika semua jalur sasaran ditemukan, daftar path sasaran akan kosong dan daftar jalur tertutup akan berisi semua jalur sasaran bersama dengan data pengujian yang sesuai mereka.   
Berikut sub-bagian membahas hasil eksperimen. Kami mengatur diskusi per eksperimen, yaitu per SUT.

*5.4.1. Pencarian biner (ns)*Hampir semua (6,96 dari 7, rata-rata) jalur sasaran layak ditemukan oleh semua kandidat selama percobaan ini.   
Semua jalur sasaran yang ditemukan selama 2 pertama (atau 1,25 rata-rata) generasi. Hasil ini didasarkan pada rata-rata 10 berjalan. Selain itu, dalam beberapa berjalan, semua jalur target layak ditemukan di pertama (yaitu, awal) generasi.   
Jelas, perilaku ini disebabkan eksplorasi dicapai oleh penduduk acak dikembangkan pada generasi awal.   
*5.4.2. Insertion sort (adalah)*Selama percobaan ini, sebagai rata-rata 10 berjalan, hampir semua (3,7 dari 4 rata-rata) jalur sasaran layak ditemukan dalam waktu 3 (atau 1,475 pada rata-rata) generasi. Dalam beberapa berjalan, semua jalur target layak ditemukan di pertama (awal) generasi. Penjelasan yang sama dengan *ns* berlaku.   
*5.4.3. Segitiga (tr)*Dalam SUT klasifikasi segitiga, semua calon ditemukan semua (4 dari 4 rata-rata) jalur sasaran layak ditemukan dalam waktu tidak lebih dari 10 (atau 7,6 rata-rata) generasi; menurut percobaan 10-run.   
*5.4.4. Minimaxi-f (mm-f)*Berkenaan dengan minimaxi-f SUT, hampir semua (12,6 dari 13 pada rata-rata) jalur sasaran layak ditemukan dalam waktu tidak lebih dari 14 (atau 9,7 rata-rata) generasi lebih dari 20-run percobaan (lihat Gambar. 3 ).   
*5.4.5. Minimaxi-i (mm-i)*Dalam percobaan ini, semua calon mampu menemukan hampir semua (12,52 dari 13 rata-rata) jalur sasaran layak ditemukan dalam waktu tidak lebih dari 36 (atau 15,56 rata-rata) generasi; dalam percobaan 20-run (lihat Gambar. 4 ). Namun, dari *phi* grafik jangka 17 (sewenang-wenang dipilih), kita dapat melihat bahwa beberapa kandidat benar-benar melakukan lebih eksploitasi (baris stabil) dari ruang pencarian sementara yang lain melakukan eksplorasi lebih (baris fluktuatif) (lihat Gambar. 5 ) . Selain itu, grafik fitness terbaik menunjukkan bahwa, untuk jangka 17 beberapa individu terbaik dari beberapa calon memang dipengaruhi oleh individu lain dalam populasi (garis fluktuatif), yaitu tidak hanya dipengaruhi oleh target paths2 (Gbr. 6 ). Garis-garis yang berfluktuasi dalam hal ini menunjukkan bahwa fitness yang “terbaik” individu mungkin drop dari satu generasi ke generasi berikutnya karena kompetisi dengan individu lainnya, dan / atau penghapusan jalur tertutup-sudah menargetkan dari set target.   
Sebuah penjelasan yang lebih rinci disediakan dalam Hermadi [8] .

*5.4.6. Bubble sort (bs)*Dalam bubble sort, hampir semua (2,98 dari 3, rata-rata) jalur sasaran layak ditemukan dalam waktu tidak lebih dari 2 (atau 1,06 rata-rata) generasi oleh semua kandidat di 10-menjalankan percobaan. Dalam hal ini, kita tidak bisa benar-benar melihat kontribusi masing-masing calon ke arah menemukan jalur sasaran, karena sebagian besar jalur sasaran ditemukan secara kebetulan di dua generasi pertama. Jelas, perilaku ini disebabkan eksplorasi dicapai oleh penduduk acak dikembangkan pada generasi awal.   
*5.4.7. Minimaxi-tri (mm-t)*minimaxi-tri adalah SUT paling menantang di antara set kita digunakan dalam eksperimen kami; ini adalah karena itu adalah kombinasi dari *mm* dan *tr* , dan memiliki sejumlah besar jalur sasaran tidak layak. Dalam eksperimen ini, hampir semua (19,3 dari 20, rata-rata) jalur sasaran layak ditemukan dalam waktu 30 (atau 19,4 rata-rata) generasi di 20-menjalankan percobaan (lihat Gambar. 7 ). Namun, dari *phi* grafik dalam jangka tertentu (9 run; sewenang-wenang dipilih), kita dapat melihat bahwa beberapa kandidat mengeksploitasi ruang pencarian lebih dari orang lain (lihat Gambar 8. Dan 9 ). Penjelasan lebih rinci disajikan dalam Hermadi [8]

*5.5.* *Analisis hasil*Secara umum, fungsi fitness calon kami menunjukkan untuk menjadi efektif dan efisien dalam menangani jalur sasaran layak yang diperlukan, terlepas dari keberadaan jalan tidak layak, panjang jalur, dan senyawa predikat kompleksitas.   
Keberadaan jalur tidak layak, jika ada, tidak menghalangi generator data uji untuk menemukan semua diberikan jalur sasaran layak melainkan membantu dalam mengeksplorasi ruang pencarian, yaitu memaksa GA untuk mereproduksi masukan yang lebih eksploratif. Pada kasus ini,   
calon yang mempekerjakan skema menguntungkan tampaknya lebih efektif dalam mengeksplorasi ruang pencarian.   
Secara umum, kandidat predikat-bijaksana sedikit lebih efektif daripada yang jalan-bijaksana, sementara calon jalur-bijaksana lebih efisien daripada yang predikat-bijaksana.

Rata-rata, fungsi fitness yang memanfaatkan pengaruh lingkungan lebih efektif daripada sebaliknya, tapi tidak lebih efisien karena waktu komputasi yang lebih.   
Umumnya, calon menerapkan skema bermanfaat dan / atau pembobotan berdasarkan-pelanggaran lebih eksploitatif daripada rekan-rekan mereka.   
Biasanya, banyak jalan sasaran dipenuhi oleh individu dalam generasi pertama. Hal ini disebabkan set awal jalur sasaran yang relatif besar, dikombinasikan dengan eksplorasi dicapai oleh pemilihan acak dari populasi awal.   
Kemudian, set jalur sasaran menjadi lebih kecil sebagai jalur sasaran sebelumnya puas dikeluarkan dari set. Misalnya di bubble sort ( *bs* ), hampir semua (2,98 dari 3) jalur sasaran layak ditemukan dalam dua generasi pertama, yang berarti bahwa jalur ini mudah untuk menemukan secara acak.   
Peningkatan jumlah jalur sasaran, terutama yang tidak layak, meningkatkan waktu perhitungan, karena kompleksitas perhitungan calon sebanding dengan jumlah jalur sasaran (misalnya, lihat *mm-f* dan *mm-i* , lihat Gambar . 3 dan 4 , masing-masing)

Jenis predikat mempengaruhi kemajuan pencarian; predikat terdiri (misalnya, lihat *tr* dan *mm-t* ) dengan operator logika AND dan predikat yang melibatkan operator relasional kesetaraan lebih sulit untuk memecahkan dan cenderung menghasilkan kurangnya lebih tinggi dari kemajuan dalam pencarian.   
Cabang lebih dalam melalui jalan yang sulit untuk memuaskan. Jalur lagi memiliki lebih kendala untuk memenuhi (misalnya, mengacu pada jalur target untuk *mm-t* ) Oleh karena itu, lebih banyak waktu komputasi yang dibutuhkan untuk menghasilkan input data untuk jenis jalur (lihat Gambar. 2 )

**6. Perbandingan dengan pekerjaan lain**Pada set perbandingan yang disajikan dalam bagian ini, kita menggunakan fungsi fitness nomor 30; yang dianggap menjadi yang terbaik pada rata-rata di semua SUTs; dalam hal efektivitas dan efisiensi.   
Dalam karya Lin [11] , yang “sama sisi” Target jalan dari program segitiga classifier dipilih untuk menunjukkan kemampuan mencari kasus uji untuk jalur tertentu dengan menggunakan GA dibandingkan dengan pengujian acak. Oleh karena itu, kami menggunakan jalur target yang sama, dan CFG untuk membandingkan pekerjaan Lin dengan kita.   
Sebenarnya, jalan target yang mengarah ke segitiga sama sisi adalah jalan yang paling sulit untuk menutupi oleh pengujian acak [11] , karena jalan tertutup jika dan hanya jika tiga parameter masukan positif dan sama. Probabilitas acak meliputi jalan ini adalah 2 - 30 (yaitu *(* 215 \* 1 \* 1 *) / (* 215 \* 215 \* 215 *)* di mana setiap bilangan bulat positif adalah 15 bit). Dengan demikian, berdasarkan teori probabilitas, itu akan mengambil pengujian acak 230 kasus uji untuk mencapai target. Generator data uji Lin mampu menutupi jalan target setelah 10 generasi, dengan 1000 individu masing-masing; yang total 10 1003 data uji rata-rata. Generator kami, bagaimanapun, adalah mampu menutupi jalan target hanya menggunakan 180 kasus uji (yaitu empat generasi, dengan 30 individu masing-masing), pada rata-rata empat berjalan. Selain itu, karena pendekatan Lin bekerja pada jalur target tunggal pada suatu waktu, kami melakukan percobaan memiliki satu jalan target pada suatu waktu. Menggunakan generator pengujian kami, dibutuhkan hanya 180 kasus uji (enam generasi dengan 30 individu, yaitu uji kasus, masing-masing) rata-rata untuk menemukan jalan target yang diperlukan, serta jalur sasaran lainnya (lihat Tabel 5 ). Dalam karya Pei [18] , ada 21 sasaran jalur, di mana delapan dari mereka adalah tidak layak, dalam pengujian (minimum-maksimum *mm* ) program. Di antara jalur sasaran layak, tiga jalur terakhir adalah yang paling sulit untuk menutupi seperti yang dilaporkan dalam karyanya. Dengan demikian, kita menggunakan jalur sasaran ini untuk perbandingan (Tabel 6 ).

Hasil di Tabel 6 menunjukkan bahwa pembangkit tes Pei mencakup tiga jalur sasaran dalam waktu 15, 2, dan 14 generasi rata-rata, masing-masing. Sementara dengan pembangkit uji kami, rata-rata tiga kali menunjukkan bahwa jalur sasaran masing-masing tertutup dalam 3, 2, dan 8 generasi dengan ukuran populasi yang lebih kecil. Dengan kata lain, pembangkit uji data kami diperiksa jumlah kurang dari individu (yaitu, data uji mungkin) untuk menutupi jalur target yang sama yang Pei tertutup.   
Harap dicatat bahwa karena pendekatan Pei bekerja pada jalur target tunggal pada suatu waktu. Untuk perbandingan yang adil, kami melakukan percobaan memiliki satu jalan target pada suatu waktu. Kami juga telah melakukan percobaan lain yang memiliki tiga sasaran jalur pada satu waktu. Hasilnya ditunjukkan pada Tabel 6 .   
Row ada. 4 dari tabel menunjukkan pembangkit kami bisa menemukan semua dari mereka dalam tujuh generasi berdasarkan rata-rata dari tiga kali. Hal ini juga diperhatikan bahwa Tabel 6 menunjukkan bahwa jumlah generasi yang diperlukan untuk menutup jalur tertentu tergantung pada tingkat kesulitan; sejak Pei melaporkan jalan yang 3 adalah yang paling sulit [18] .   
Tabel 7 menggambarkan hasil rinci dari berbeda berjalan generator pengujian kami, lebih dari 20 berjalan; mencoba untuk menutupi tiga jalur target yang paling sulit dalam pekerjaan Pei baik oleh jalur tunggal atau semua jalur pada suatu waktu. Tabel ini dimaksudkan untuk menunjukkan konsistensi hasil.   
Rata-rata, populasi yang lebih besar ukuran jumlah yang lebih kecil dari generasi yang diperlukan untuk menemukan jalan target (s).   
Tabel 8 , di bawah, menunjukkan hasil generator kami, berdasarkan 20 berjalan, mencoba untuk menutupi hanya “sama sisi” jalan target. Tabel tersebut juga menunjukkan hasil ketika mencoba untuk memenuhi semua jalur target pada suatu waktu.   
Rata-rata, semakin besar populasi ukuran jumlah yang lebih kecil dari generasi yang diperlukan untuk menemukan jalan target (s), yang juga didukung oleh standar deviasi yang lebih kecil.   
**7. Kesimpulan dan pekerjaan di masa depan**Dalam tulisan ini, kami telah menyajikan tes berbasis GA Generator data yang mampu menghasilkan beberapa data uji untuk menutupi beberapa jalur target dalam satu run. Kami telah menunjukkan kemampuan pendekatan yang diusulkan melalui validasi empiris, dan dibandingkan sejumlah variasi generator yang diusulkan.   
Hasil percobaan menunjukkan bahwa pembangkit uji data kami ismore efektif andmore efisien dibandingkan generator yang ada; karena fakta bahwa hal itu memungkinkan meliputi beberapa jalur target dengan jumlah kurang dari data uji diperiksa.   
Namun, berikut ini adalah kekhawatiran bahwa membatasi skalabilitas generator dan kegunaan:   
1. konstruksi CFG manual membutuhkan cukup banyak waktu untuk melakukannya.   
2. Manual identifikasi sasaran jalur membutuhkan kreativitas tester.   
3. Proses Program instrumentasi manual bahasa pemrograman tergantung, yang harus dilakukan dengan hati-hati sehingga tidak mengubah semantik dari program. Oleh karena itu, instrumentasi manual membutuhkan sejumlah besar pekerjaan dan waktu.   
4. Berkenaan dengan predikat-bijaksana traversal, fungsi fitness kami tidak menganggap cocok dengan *sub-jalur* yang memiliki posisi yang tak tertandingi untuk kemungkinan kontribusi positif terhadap nilai fitness. Hanya menganggap mereka sub-jalur yang memiliki posisi yang sama.   
Pekerjaan di masa depan akan mencoba untuk mengatasi keterbatasan di atas. Selain itu, kami juga akan mencoba untuk menyelidiki kemampuan untuk memungkinkan identifikasi otomatis potensi jalur Program tidak layak. Pengujian perangkat lunak berorientasi objek akan tujuan lain dari penelitian masa depan kita.  
**Ucapan Terima Kasih** Para penulis ingin mengakui King Fahd University of Petroleum and Minerals (KFUPM) untuk memanfaatkan berbagai fasilitas dalam melaksanakan penelitian ini. Banyak terima kasih karena wasit anonim untuk rinci dan membantu komentar mereka.