Biasanya, mencari datum masukan di kolam (domain / set) dari data masukan yang mungkin ditangani sebagai masalah pengoptimalan [19]. Pada usia dini otomatisasi pengujian perangkat lunak, sebagian besar generator data uji menggunakan algoritma gradien keturunan. Namun, algoritma ini tidak efisien, dan memakan waktu, dan tidak dapat melepaskan diri dari optima lokal di ruang pencarian domain dari data masukan yang mungkin [25]. Isu-isu ini memerlukan kebutuhan untuk menyelidiki kesesuaian algoritma pencarian meta-heuristik, mis. Simulasi anil, algoritma genetika, dan optimasi koloni semut sebagai alternatif yang lebih baik untuk mengembangkan generator data uji. Namun, sampai pengetahuan kami, sejauh ini, peneliti hanya tertarik menggunakan algoritma genetika untuk menghasilkan data uji [43].  
Wegener dkk. Telah menunjukkan kesesuaian menggunakan algoritma evolusioner dalam pengujian perangkat lunak [43]. Dengan menggunakan perhitungan evolusioner, para periset telah melakukan beberapa pekerjaan dalam mengembangkan generator data uji genetika berbasis algoritma genetika [6] [43]. Namun, dalam mencoba untuk mengidentifikasi kekuatan dan kelemahan dari berbagai teknik yang tersedia untuk pengembangan generator data uji yang ada, kami menemukan bahwa tidak ada atribut kualitas atribut yang dapat didefinisikan dengan baik yang dapat digunakan untuk membandingkan berbagai teknik tersebut.  
Selain itu, seperti yang dibahas pada Bab 3, salah satu pengamatan kami terhadap generator data uji berbasis GA yang ada adalah bahwa mereka dapat menghasilkan hanya satu datum uji pada satu waktu. Oleh karena itu, dalam usaha untuk menghasilkan satu set data uji (yaitu, lebih dari satu datum uji) untuk memenuhi kriteria tertentu (disebut kriteria kecukupan uji, misalnya cakupan cabang) yang dipertimbangkan, generator uji harus digunakan lebih dari satu kali (satu putaran 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 generator data uji yang ada tidak efisien dalam mencoba menghasilkan banyak data uji. Survei kritis terperinci dan tinjauan terhadap pendekatan yang ada ditunjukkan pada Bab 3.  
Motivasi kami untuk menyelidiki masalah ini berasal dari manfaat yang di atas.

**Kontribusi utama karya tesis ini adalah sebagai berikut:**  
¾ Mengusulkan seperangkat atribut untuk menilai dan membandingkan generator data uji berbasis GA;  
¾ Membandingkan generator data uji berbasis GA yang tersedia berdasarkan rangkaian atribut yang diusulkan;  
¾ Mengusulkan generator data uji berbasis GA yang mampu menghasilkan banyak  
Uji data untuk melihat beberapa jalur target1 pada satu run;  
¾ Melaksanakan dan membandingkan sejumlah variasi generator yang diusulkan2;  
¾ Melakukan percobaan untuk menunjukkan kekuatan pendekatan yang diusulkan menggunakan Matlab

2.1. pengantar  
Bab ini membahas teknik pengujian perangkat lunak dan kriteria kecocokan uji yang berbeda. Bab ini juga memberikan latar belakang yang diperlukan tentang bagaimana Algoritma Genetika bekerja.  
2.2. Teknik Pengujian Perangkat Lunak  
Umumnya, teknik pengujian perangkat lunak dikelompokkan menjadi dua kategori: analisis statis dan pengujian dinamis [18] [33] [38]. Dalam analisis statis, pengkodean kode membaca kode sumber program, pernyataan berdasarkan pernyataan, dan secara visual mengikuti alur program logis dengan memberi masukan. Jenis pengujian ini sangat bergantung pada pengalaman resensi. Analisis statis menggunakan persyaratan program dan dokumen desain untuk tinjauan visual. Sebaliknya, teknik pengujian dinamis menjalankan program yang diuji pada data masukan uji dan mengamati hasilnya. Biasanya, istilah pengujian mengacu hanya pada pengujian dinamis.  
Subbagian berikut memberikan latar belakang singkat pada kedua kategori pengujian ini.  
2.2.1. Analisis Statis  
Selama bertahun-tahun, sebagian besar programmer menganggap bahwa program tersebut ditulis semata-mata untuk eksekusi mesin dan tidak dimaksudkan untuk dibaca oleh manusia, dan satu-satunya cara untuk menguji sebuah program adalah dengan menjalankannya di mesin. Cara ini mulai berubah pada awal 1970-an, karena karya Weinberg tentang "The Psychology of Computer Programming" [46]. Weinberg memberikan argumen yang meyakinkan mengapa program harus dibaca oleh orang-orang dan mengindikasikan bahwa ini bisa menjadi proses pendeteksian kesalahan yang efektif.  
Pengalaman menunjukkan bahwa analisis statis, a.k.a. pengujian non-komputer berbasis atau manusia,  
Metode cukup efektif dalam menemukan kesalahan [33]. Metode analisis statis dimaksudkan untuk diterapkan selama periode antara penyelesaian kode dan awal pengujian berbasis eksekusi.  
Metode analisis statis yang umum adalah pemeriksaan kode, panduan kode, pengecekan meja, dan ulasan kode [33]. Inspeksi kode dan panduan adalah dua metode analisis statis utama dan mereka memiliki banyak kesamaan. Inspeksi dan penelusuran mencakup pemeriksaan bacaan atau inspeksi visual oleh sekelompok orang. Kedua metode tersebut melibatkan beberapa karya persiapan oleh para peserta. Klimaks adalah pertemuan pikiran, mis.  
Brainstorming, dalam pertemuan seperti konferensi yang diadakan oleh para peserta. Tujuan pertemuan tersebut adalah untuk menemukan kesalahan, namun tidak menemukan solusi atas kesalahan, yaitu menguji tetapi tidak melakukan debug.8  
2.2.1.1. Pemeriksaan Kode  
Pemeriksaan kode adalah seperangkat prosedur dan teknik deteksi kesalahan untuk pembacaan kode kelompok. Sebagian besar diskusi tentang inspeksi kode fokus pada prosedur, formulir yang harus diisi, dan sebagainya.  
Selama sesi pemeriksaan, dua kegiatan dilakukan: narasi kode dan pemeriksaan kode. Kode dibaca pernyataan oleh pernyataan dan dianalisis sehubungan dengan daftar kesalahan historis pemrograman umum (misalnya data-reference, data-declaration, computation, comparison, control-flow, input / output, interface).  
2.2.1.2. Kode Walkthroughs  
Prosedur awalnya identik dengan proses pemeriksaan. Perbedaannya, bagaimanapun, adalah karena alih-alih hanya membaca program atau menggunakan daftar periksa kesalahan, salah satu peserta yang ditunjuk sebagai penguji datang ke pertemuan dengan sejumlah kecil kasus uji kertas yang mewakili rangkaian input dan keluaran yang diharapkan untuk tes Program atau modul  
Selama pertemuan, setiap ujian dinyatakan secara mental, yaitu data uji berjalan melalui logika program. Keadaan program, yaitu nilai variabel, dipantau di atas kertas atau papan tulis.  
Jelas, kasus uji harus sederhana di alam dan sedikit jumlahnya, karena orang melakukan program pada tingkat yang jauh lebih lambat daripada mesin. Dengan demikian, kasus uji coba sendiri tidak memainkan peran penting; Sebaliknya, mereka berfungsi sebagai wahana untuk memulai dan untuk menanyai pemrogram tentang logika dan anggapannya. Dalam kebanyakan walkthrough, lebih banyak kesalahan ditemukan selama proses tanya jawab programmer daripada yang ditemukan langsung oleh test case itu sendiri.  
2.2.1.3. Pemeriksa meja  
Pengecekan meja dapat dilihat sebagai inspeksi atau penelusuran satu orang; Seseorang membaca sebuah program, memeriksanya sehubungan dengan daftar kesalahan, dan / atau menempuh data uji melaluinya.  
Ada tiga alasan utama mengapa pengecekan meja, bagi kebanyakan orang, relatif tidak produktif: proses yang benar-benar tidak disiplin, prinsip bahwa orang pada umumnya tidak efektif dalam menguji program mereka, dan tidak ada kompetisi seperti dalam kerja tim.  
2.2.1.4. Ulasan Kode (Peringkat rekan)  
Kajian kode adalah teknik untuk mengevaluasi program anonim dalam hal kualitas, rawatan, perluasan, kegunaan, dan kejelasan secara keseluruhan. Tujuan dari tinjauan ini adalah untuk memberikan penilaian pemrogram. Sekelompok pemrogram diberi beberapa program terpilih untuk menilai berdasarkan skala tertentu yang tertulis dalam formulir tinjauan.