ALGORITMA GENETIKA

Teori dan Aplikasinya untuk **Bisnis dan Industri**

Teori dan aplikasi algoritma genetika berkembang pesat dan merambah di berbagai aplikasi dunia nyata pada berbagai bidang, termasuk komputasi, transportasi, teknik, kedokteran, pertanian, industri, bio-kimia, dan robotika. Algoritma genetika bersifat evolusionari, stokastik, heuristik dan progresif mengantarkan ke solusi problema optimal dengan prinsip seleksi alam dan kelangsungan hidup (*survival*) yang terarah dan terkendali.

Dengan berlimpah manfaat dan aplikasinya di dunia nyata, algoritma genetika hanyalah pengetahuan bak sebutir debu bahkan lebih kecil dibandingkan kumpulan ilmu Sang Pencipta yang diturunkan kepada kita. Dengan berbagai objek dan fenomena alam yang penuh dengan aneka pengetahuan, kita hanya mengolah dan memanfaatkan ulang pengetahuan tersebut. Prinsip genetika yang terkait erat dengan kromosom yang berisi *gen* adalah kunci utama dalam memahami terjadinya pelestarian sifat dari tetua makhluk ke generasi (turunan) berikutnya. Prinsip ini ternyata berkontribusi besar dalam bidang komputasi cerdas (*intelligent computing*) yang terus berkembang agresif. Buku ini menyajikan pengetahuan mengenai algoritma genetika yang mudah dipahami karena dilengkapi dengan contoh-contoh kasus.

PT Penerbit IPB Press
Kampus IPB Taman Kencana
Jl. Taman Kencana No. 3, Bogor 16128
Telp. 0251 - 8355 158 E-mail: ipbpress@ymail.com

Sains
ISBN :978-979-493-437-1
9 789794 934371

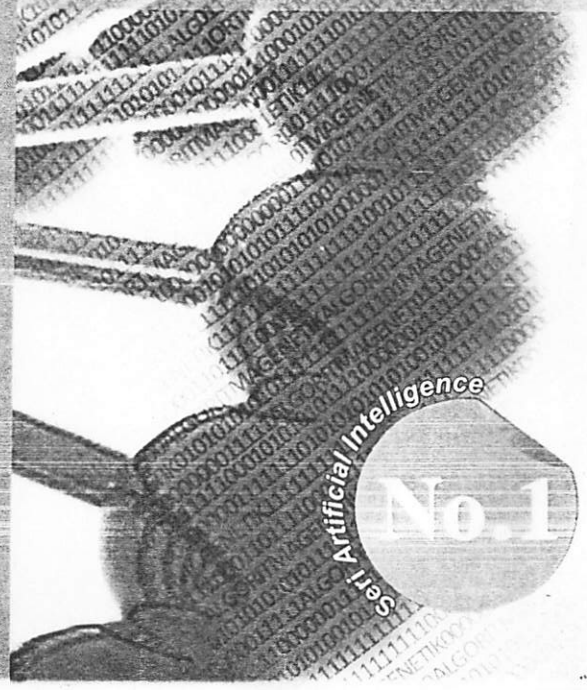
ALGORITMA GENETIKA | Teori dan Aplikasinya untuk Bisnis dan Industri

IPB Press

ALGORITMA GENETIKA
Teori dan Aplikasinya untuk Bisnis dan Industri

IPB Press

Yandra Arkeman
Kudang Boro Seminar
Hendra Gunawan



10/10/12
YANDRA ARICEMAN

ALGORITMA GENETIKA

Teori dan Aplikasinya untuk Bisnis dan Industri

ALGORITMA GENETIKA

Teori dan Aplikasinya untuk Bisnis dan Industri

Yandra Arkeman

Kudang Boro Seminar

Hendra Gunawan

IPB Press

ALGORITMA GENETIKA

Teori dan Aplikasinya untuk Bisnis dan Industri

Yandra Arkeman

Kudang Boro Seminar

Hendra Gunawan

Penyunting Bahasa : Elviana
Korektor Bahasa : Hans Baihaqi
Desainer Sampul & Tata Letak : Sani Etyarsah
Sumber gambar cover : sepocikopi.com

PT Penerbit IPB Press
Kampus IPB Taman Kencana Bogor
Cetakan Pertama: September 2012

ISBN: 978-979-493-437-1

Kata Pengantar

Teori dan aplikasi algoritma genetika berkembang pesat dan merambah di berbagai aplikasi dunia nyata di berbagai bidang termasuk komputasi, transportasi, teknik, kedokteran, pertanian, industri, biokimia, dan robotika. Algoritma genetika bersifat evolusionari, stokastik, heuristik dan progresif mengantarkan ke solusi problema optimal dengan prinsip seleksi alam dan kelangsungan hidup (*survival*) yang terarah dan terkendali.

Dengan berlimpah manfaat dan aplikasinya di dunia nyata, algoritma genetika hanyalah pengetahuan bak sebutir debu bahkan lebih kecil dibandingkan kumpulan ilmu ALLAH yang diturunkan kepada kita. Kita diilhami dengan berbagai objek dan fenomena alam yang penuh dengan aneka pengetahuan yg hakikinya kita hanya mengolah dan memanfaatkan ulang pengetahuan tersebut. Prinsip genetika yang terkait erat dengan kromosom yang berisi gen adalah kunci utama dalam memahami terjadinya pelestarian sifat dari tetua makhluk ke generasi (turunan) berikutnya. Prinsip ini ternyata berkontribusi besar dalam bidang komputasi cerdas (*intelligent computing*) yang terus berkembang agresif.

Kami sungguh bersyukur kehadirat ALLAH Tuhan Yang Maha Berilmu atas karunia dan ijin-NYA dapat menulis dan berbagi pengetahuan tentang teori dan aplikasi algoritma genetika dengan segala keterbatasan kami sebagai manusia biasa. Tiada ilmu bagi kita kecuali apa yang telah diajarkan-NYA kepada kita. Kami berterima-kasih kepada Nurul Khairani, Gibtha Fitri Laxmi, Resa Denasta Syarif, penerbit IPB Press dan semua pihak yang telah ikut membantu terselesaikannya dan terpublikasikannya buku ini. Terima kasih juga kami ucapkan

kepada Mushthofa, S.Kom, M.Sc atas kesediaannya me-*review* naskah buku ini secara menyeluruh.

Akhir kata, kami sangat mengharap dan menyambut sumbangan ide, saran, dan kritik untuk menyempurnakan buku ini ke depan agar lebih bermakna dan bermanfaat.

Bogor, 23 September 2012

YA, KBS, HG

Daftar Isi

Bab 1
Pendahuluan: *State Of The Art Of Intelligent Systems*

Salah satu cabang ilmu komputer (*computer science*) yang berkembang sangat pesat dewasa ini adalah *artificial intelligence* (AI) atau kecerdasan buatan. Sejak kelahirannya sampai saat ini, cabang ilmu baru ini sudah banyak ditekuni oleh para ilmuwan, peneliti, dan praktisi. Selain itu, hasilnya pun telah banyak diterapkan secara komersil di berbagai bidang kehidupan. Sebagai contoh, pada alat-alat komersil seperti mesin cuci, kamera, AC, kulkas, kini sudah diberi label "pintar" atau *intelligent* yang menandakan alat tersebut sudah mengadopsi teknologi pintar, yang membuat konsumen semakin nyaman dan mudah dalam pengoperasiannya. Untuk selanjutnya, peralatan-peralatan pintar semacam ini akan semakin banyak lagi kalau ditambahkan lagi dengan bidang ilmu dan teknologi maju lainnya, seperti lengan robot otomatis di *space-shuttle* yang dibuat NASA dan robot yang dipakai untuk ekspedisi di planet Mars. Selain itu, AI banyak pula digunakan di dalam bisnis dan industri untuk membantu manajer dalam proses pengambilan keputusan (*decision making*). Alat bantu (*tool*) untuk para manajer ini sering disebut *Intelligent Decision Support Systems* (IDSS) atau *Intelligent Business Systems* (IBS). Dengan menggunakan alat bantu yang "cerdas", tentu saja keputusan yang diambil oleh manajer akan lebih akurat dan lebih cepat dibandingkan dengan alat-alat atau metode-metode konvensional yang sudah pernah ada sebelumnya.

Usaha manusia untuk membuat mesin pintar (*intelligent machine*) ini memang sudah dimulai sejak puluhan tahun yang lalu,

yaitu pada saat komputer diciptakan dan mulai digunakan secara luas di banyak kehidupan manusia. Pada saat itu, timbul keinginan manusia untuk mengubah komputer dari yang hanya sekedar alat pengolah data (*data processing*) menjadi pengolah informasi (*information processing*). Kemampuan mengolah informasi inilah yang dipercaya sebagai salah satu kemampuan manusia yang perlu ditiru oleh komputer sehingga komputer tidak hanya sekedar dapat berhitung (*to compute*) tetapi juga dapat "berpikir" (*to think*) dan membuat keputusan-keputusan sulit layaknya manusia pada umumnya atau para ahli (*experts/specialists*) pada khususnya.

Teknik dalam AI telah banyak dikembangkan dalam berbagai bidang, teknik yang biasa dikembangkan yaitu :(1) Sistem Pakar (*Expert systems*), (2) *Fuzzy Logic*, (3) Jaringan Saraf Tiruan (*Artificial Neural Networks*), dan (4) Algoritma Genetika (*Genetic Algorithms*). Sebelum membahas secara mendalam tentang Algoritma genetika yang menjadi topik buku ini, keempat teknik AI tersebut akan diuraikan secara singkat dan jelas berikut ini.

1.1 Sistem Pakar (*Expert System*)

Sistem pakar adalah sistem komputer yang dibuat untuk meniru cara manusia dalam proses menarik suatu kesimpulan (*reasoning*) ketika hendak menyelesaikan suatu masalah. Dalam proses pemecahan masalah ini, manusia biasanya menggunakan fakta dan aturan-aturan *heuristik* yang didapatkan dari pengetahuan, pengalaman, atau dari keahlian yang dimilikinya. Dalam sistem pakar itu, fakta dan aturan ini disimpan dalam komponen yang disebut basis pengetahuan (*knowledge base*). Selain basis pengetahuan, sistem pakar juga memiliki komponen lain yaitu mesin inferensi (*inference engine*). Mesin inferensi adalah sebuah program komputer yang bertugas untuk membuat inferensi (penarikan kesimpulan) berdasarkan fakta-fakta dan aturan-aturan yang disimpan dalam *knowledge-base*. Kesimpulan atau keputusan yang

dibuat bisa bersifat induksi (*inductive reasoning*) atau deduksi (*deductive reasoning*). Dalam proses membuat kesimpulan itu, mesin inferensi bisa menggunakan dua pendekatan yaitu *forward-chaining* dan *backward chaining*.

Sebagai contoh yaitu bagaimana seorang penerjun payung bekerja. Ketika kecepatan angin tinggi, ia akan menarik payungnya lebih kuat. Sebaliknya, kalau angin tidak kencang, tali payung bisa lebih dilonggarkan. Selain itu, banyak fakta dan aturan lain yang harus ia ikuti sehingga ia dapat mendapat dengan selamat di tempat yang dituju. Fakta dan aturan tersebut (yang biasanya dinyatakan dalam bentuk *IF-THEN* atau JIKA-MAKA) didapatkan dari pengalaman dan keahliannya sebagai penerjun. Perhatikan juga bahwa si penerjun bekerja dengan fakta yang berbentuk simbol (contohnya: kecepatan angin tinggi) dan aturan yang tidak terstruktur (banyak aturan dan masing-masing aturan saling terkait satu sama lain).

Dalam sebuah sistem pakar, fakta, dan aturan yang diperoleh dari para pakar (dan juga diperoleh dari sumber lain) disimpan di dalam *knowledge-base* untuk kemudian digunakan oleh *inference engine* dalam pengambilan keputusan. Dengan cara seperti ini, keahlian seseorang dapat digunakan oleh orang lain dengan perantaraan komputer. Selain itu, *kowledge-base* dari sistem pakar juga dapat diperbarui secara berkala sehingga sistem tersebut dari hari ke hari bisa "belajar" dan menjadi semakin "pintar".

Sistem pakar telah dibuat untuk beberapa bidang aplikasi. Contohnya MYCIN, sebuah sistem pakar komersial bidang kedokteran yang dibuat tahun 70-an oleh Ed Shortliffe (dari Stanford University), sukses digunakan untuk mendiagnosis penyakit akibat infeksi darah. Demikian pula INTERNIST, sebuah sistem yang dibuat untuk membantu dokter mendiagnosis berbagai penyakit dalam. Di bidang teknolgi informasi, dibuat XCON yang mampu memecahkan masalah pengaturan

konfigurasi komputer yang rumit. Sementara itu, di bidang bisnis dan manajemen, sistem pakar mempunyai prospek yang sangat baik untuk digunakan. Suatu contoh sederhana adalah sistem pakar dapat digunakan untuk mendiagnosis kesehatan perusahaan atau menilai kelayakan finansial suatu bisnis baru.

1.2 Logika *Fuzzy* (*Fuzzy Logic*)

Sebelum ditemukannya logika *fuzzy* (*fuzzy logic*), logika yang kita gunakan sebelumnya (baik untuk bidang keteknikan atau pun untuk pengambilan keputusan manajerial) adalah logika *Boolean* (*Boolean logic*) yang disimbolkan dengan angka 0-1, *on-off, yes-no, go-nogo*, dan sebagainya. Berbeda dengan logika *Boolean* yang merepresentasikan keadaan hanya dalam dua status (hidup-mati, hitam-putih, 0-1, dan sebagainya), logika *fuzzy* dapat merepresentasikan keadaan lebih dari dua status. Misalnya, antara 0 dan 1 ada setengah, seperempat, sepertiga, dan sebagainya. Antara *yes* dan *no* ada setengah *yes*, sepertiga *yes*, seperlima *yes* dan sebagainya. Antara hitam dan putih, ada abu-abu!! (Ingat kata *fuzzy* berarti samar, tidak hitam, dan tidak pula putih).

Lalu, mengapa kita harus menggunakan logika *fuzzy*? Jawabnya adalah karena banyak persoalan yang ada di alam ini memiliki fakta atau kondisi yang samar. Sebagai contoh, kita sering menggunakan pernyataan orang yang *kuat*, manusia yang *cerdas*, buah yang *hijau*, perusahaan yang *besar*, kecepatan angin *tinggi*, dan sebagainya, untuk mengungkapkan suatu persoalan atau mengategorikan sesuatu hal. Kata *kuat, cerdas, hijau, besar,* dan *tinggi* adalah bentuk-bentuk informasi yang *ambigious, fuzzy,* atau samar. Misalnya orang yang cerdas kriterianya adalah IQ-nya di atas 110. Kalau ada orang IQ-nya 130, sudah jelas dia cerdas. Kalau IQ-nya 90 sudah jelas dia tidak cerdas. Akan tetapi bagaimana kalau orang IQ-nya 109? Cerdaskah dia? Apa bedanya dengan orang yang ber-IQ 111?

Dalam kasus seperti di atas, logika *fuzzy* mampu melakukan *reasoning*, karena penggolongannya bersifat kontinu (*gradations*) dan tidak bersifat *crisp* (*sharp distinctions*) seperti halnya logika *Boolean*. Hasil penelitian para ahli menunjukkan bahwa sistem syaraf otak dan panca indera manusia tidak bekerja dengan logika *Boolean*, tetapi dengan logika lain (yang belum diketahui?!). Logika "lain" inilah yang coba didekati oleh logika *fuzzy*.

Sebagai ilustrasi, kita kembali ke cerita si penerjun payung. Apa yang dia lakukan jika kecepatan angin *cukup tinggi* dan jarak dari permukaan tanah *cukup dekat*? Jawabnya adalah "Tarik tali payung *agak kuat* !". Kecepatan angin yang cukup tinggi dan jarak yang agak dekat dapat diidentifikasi oleh syaraf indera dan otak si penerjun. Kemudian otak menyuruh tangannya untuk menarik tali agak kuat. Bentuk tarikan yang agak kuat ini hanya dapat diterjemahkan oleh otak si penerjun tadi. Demikian pula halnya dengan seorang masinis kereta api bawah tanah. Salah satu contoh aturan yang harus diikutinya adalah "Jika stasiun sudah *dekat*, kecepatan harus *lambat*". Istilah dekat dan lambat ini adalah istilah yang bersifat samar yang dapat diolah dengan logika *fuzzy*.

Hal yang menarik adalah bahwa sampai saat ini logika *fuzyy* sudah banyak diterapkan pada alat-alat elektronik seperti mesin cuci, kamera foto dan video, AC, dan lain-lain. Namun demikian, dalam beberapa tahun belakangan ini aplikasi logika *fuzzy* di bidang bisnis dan manajemen kian bertambah. Apa yang harus dilakukan manajer bila angka penjualan *agak rendah*, tingkat suku bunga uang *cukup tinggi*, laju inflasi *agak kecil*, dan persaingan dunia usaha *sangat ketat*? Gunakan logika *fuzzy* biar lebih mudah!

1.3 Jaringan Saraf Tiruan *(Artificial Neural Networks)*

Sesuai dengan namanya, teknologi ini dibuat untuk meniru jaringan saraf otak manusia (*neuron*). Neuron adalah sel otak manusia yang memproses data menjadi informasi. Otak manusia terdiri atas jutaan (triliunan?) sel *neuron* yang bertugas menerima rangsang dan mengolahnya menjadi informasi serta pengetahuan. Sebagai contoh, sewaktu anda melihat satu deret huruf berikut ini: "NAMA", sel-sel otak kita menerima rangsang berupa bentuk huruf seperti N, A, M, A. Kemudian keempat huruf tersebut digabung menjadi satu kata, dan selanjutnya kata itu dapat kita mengerti (karena kita sudah belajar sebelumnya tentang jenis huruf dan bagaimana cara membaca kumpulan huruf serta memahami artinya). Proses ini berjalan "sangat-sangat-cepat-sekali", karena manusia memang makhluk cerdas dengan jumlah sel otak yang sangat banyak sehingga mempunyai kemampuan belajar dan berpikir yang sangat tinggi (tapi bukan tidak terbatas). Bayangkan jika kata "NAMA" di atas anda perlihatkan pada kucing atau anjing anda di rumah. Apakah mereka akan mengerti? Seandainya pun anda mampu mengajar kucing anda membaca satu kata di atas, bisakah anda mengajar kucing anda membaca dan memahami buku yang sedang anda baca ini? Berapa lama waktu yang dibutuhkan mereka untuk belajar?

Mekanisme cara manusia belajar dan berpikir seperti ini dicoba ditiru oleh para ahli untuk membuat mesin yang bisa "belajar" dan "berpikir" seperti layaknya manusia. Hal ini bukanlah suatu keanehan, karena banyak teknologi maju yang ada sekarang ini, memang diilhami oleh fenomena alam (karya agung Sang Pencipta). Suatu contoh, struktur sayap pesawat terbang dibuat sedemikian rupa seperti struktur sayap burung. Dengan struktur seperti sayap burung tersebut, udara di atas sayap akan berjalan lebih cepat dibandingkan udara di bawahnya.

Efeknya adalah udara di atas sayap lebih "ringan" dari udara di bawah sayap, sehingga timbullah daya angkat yang bisa membuat pesawat terbang melayang di angkasa (Hukum Bernoulli).

Artificial Neural Networks atau Jaringan Saraf Tiruan (JST) terdiri atas beberapa *neuron* tiruan. Satu *neuron* tiruan (selanjutnya disebut *neuron* saja) terdiri atas beberapa jaringan *input* dan satu jaringan *output*. Setiap kelompok *neuron* dihubungkan dengan beberapa kelompok *neuron* lainnya hingga beberapa lapis (*layer*). *Input* data diterima olen beberapa *neuron* pada lapisan pertama (*input layer*), kemudian diolah oleh beberapa *neuron* pada lapisan-lapisan tersembunyi (*hidden layer*) dengan menggunakan beberapa macam **fungsi aktivasi** secara sekaligus (paralel). Selanjutnya, hasil pengolahan *hidden layer* diberikan kepada *neuron-neuron* lapisan terakhir (*output layer*). *Neuron* terakhir inilah yang memberikan informasi yang kita perlukan untuk membuat aksi, tindakan, atau pun keputusan.

Dengan struktur seperti di atas, JST mampu melakukan proses pembelajaran dengan menggunakan contoh (*learning by examples*). Sebagai contoh, di Amerika kini sedang dikembangkan sistem kemudi otomatis (*car's auto pilot systems*) dengan menggunakan JST. Pada sistem ini, JST diajarkan untuk menaikkan dan menurunkan kecepatan secara halus (*smooth*) sehingga para penumpang dapat berkendaraan secara nyaman. JST dilatih dengan memberikan contoh berapa kecepatan yang seharusnya dipakai, termasuk kecepatan mobil kita sekarang, kecepatan mobil di depan kita, jarak antara kedua mobil, kondisi jalan, dan sebagainya. Setelah proses pembelajaran dan pelatihan ini selesai, JST akan bekerja sendiri untuk mengemudikan mobil. Dengan kemajuan yang cukup sigifikan dari tahun ke tahun, suatu saat seluruh kemampuan pengemudi dapat tergantikan oleh JST ini sehingga impian manusia untuk berkendara tanpa supir atau terbang tanpa pilot akan menjadi kenyataan.

Di bidang bisnis dan industri pun JST sudah banyak digunakan. Hal ini disebabkan JST memiliki kemampuan yang sangat baik untuk mengenali atau membaca pola (*pattern recognition*). Kemampuan mengenali dan membaca pola ini dipercaya sebagai kemampuan yang hanya dimiliki oleh manusia, yaitu para spesialis di bidang terkait untuk meramalkan apa yang akan terjadi atau tindakan apa yang perlu dilakukan. Sebagai contoh, seorang ahli gunung berapi (*geologist*) setiap harinya diberi data tentang aktivitas gunung yang diawasinya. Berdasarkan data yang dicatat dari waktu ke waktu ini, si ahli tersebut dapat menentukan sifat gunung berapi yang diawasinya, apakah dalam keadaan aman atau bahaya. Hal ini dapat ia lakukan karena si ahli tersebut sudah mempelajari ilmu tentang gunung berapi. Demikian juga seorang dokter. Dia dapat mendiagnosis apakah pasiennya sehat atau punya kelainan organik berdasarkan data hasil pemeriksaan fisik dan laboratorium. Data hasil pemeriksaan fisik dan laboratorium ini bukanlah data yang dapat diinterpretasikan oleh semua orang, karena sifatnya yang sangat tidak terstruktur. Seorang dokter pun bahkan harus belajar sampai tingkat spesialis untuk memahami data ini. Demikian juga seorang pialang ahli di bursa saham, dia dapat menentukan kapan dan berapa banyak saham harus dibeli atau dijual berdasarkan pengetahuan yang dimilikinya.

Kemampuan para ahli di atas dapat diajarkan kepada JST, sehingga JST mampu memberikan informasi dan memberi rekomendasi kepada kita tentang apa yang terjadi dan apa yang harus dilakukan. Sebagai contoh jika data tentang aktivitas gunung berapi (*input-output*) selama 10 tahun terakhir kita masukkan ke dalam komputer, kemudian kita olah dengan JST, dengan mudah kita dapat segera tahu apakah gunung berapi itu dalam keadaan aman atau tidak. Hal yang perlu diperhatikan di sini adalah bahwa JST sudah kita ajar dan latih dengan cukup, sehingga JST sudah paham dengan pola kegiatan gunung berapi tersebut, sama halnya seperti seorang *geologist*.

Hal yang sama dapat dilakukan di bidang bisnis, *Customer Relationship Management* (CRM). Misalnya, dapatkah seorang analis kredit mengetahui kalau seorang yang datang padanya adalah seorang *customer* yang potensial atau tidak? Misalnya, jika selama 10 tahun terakhir ini ia mencatat bagaimana ciri atau tingkah laku *customer* yang datang itu dan kemudian juga mencatat apakah *customer* tersebut kreditnya lancar atau tidak, dengan menggunakan JST ia dapat mempelajari pola tingkah laku *customer*-nya. Kemudian, ia juga dapat segera tahu apakah *customer* yang ada dihadapannya saat itu adalah *customer* yang potensial atau tidak. Adanya proses pembelajaran pola tingkah laku dan pelatihan selama 10 tahun inilah yang dapat membuat JST bertindak sebagai ahli yang dapat meramalkan karakteristik *customer*. Seberapa besar tingkat kebenarannya? Sama seperti manusia, tingkat kepercayaan JST tidak bisa sampai 100%. Umumnya, kalau tingkat kebenaran *output* JST sudah di atas 90%, dapat dikatakan bahwa JST yang dibuat sudah layak untuk digunakan.

Sampai saat ini JST sudah banyak dibuat untuk berbagai keperluan, baik di bidang teknik, bisnis, dan industri. Sebagian proyek masih berada pada skala riset universitas dan beberapa sudah berada pada tingkat komersial di pasaran. JST dapat dibuat dengan berbagai macam jenis *software* seperti *NeuralWare*, *MatLab*, bahasa program *Pascal* atau C++. Hal yang sangat kritis dalam pembuatan JST adalah penentuan bentuk **fungsi aktivasi** dan penentuan jumlah *neuron* pada lapisan tersembunyi (*hidden layer*). Kedua hal ini akan sangat menentukan apakah proses pembelajaran JST akan "sukses" (*convergent*) atau tidak.

1.4 Algoritma Genetika (*Genetic Algorithms*)

Banyak persoalan dalam kehidupan manusia yang dikategorikan sebagai masalah "*searching*", yaitu masalah untuk mencari satu pilihan yang paling baik (paling memuaskan) di antara beberapa kemungkinan

yang ada. Suatu contoh sederhana, misalnya seseorang ingin pergi berlibur ke suatu tempat. Banyak pilihan jenis pesawat, mobil, hotel, atau restoran yang tersedia. Ia tentu saja harus memutuskan satu kombinasi dari beberapa kombinasi yang tersedia untuk memuaskan keinginannya. Kadang-kadang masalah makin dipersulit karena adanya pertimbangan lain yang perlu diperhatikan. Contohnya, pada satu sisi ia ingin menghemat uang, sedangkan pada sisi lain ia ingin penerbangan yang nyaman.

Masalah "*searching*" juga banyak dijumpai di bidang keteknikan dan manajemen. Sebagai contoh, sebuah robot yang digunakan untuk transportasi barang di dalam pabrik akan melakukan "*searching*" untuk menentukan rute yang paling baik menuju satu tempat tertentu. Contoh lain adalah menara pengatur lalu lintas penerbangan di bandara. Di menara ini, petugas dibantu oleh alat-alat berteknologi tinggi, berusaha mencari urutan pendaratan pesawat yang paling aman dan efisien dari berbagai kombinasi urutan pendaratan yang ada setiap harinya. Masalah "*searching*" juga muncul dalam permainan catur atau bentuk permainan strategi lainnya. Dalam permainan catur misalnya, seseorang akan mencari urutan langkah terbaik untuk menaklukkan lawannya secepat mungkin.

Seperti diketahui, istilah "*searching*" dilakukan untuk mencari suatu objek dalam sekumpulan objek yang ada. Dalam masalah optimasi, objek yang dicari adalah solusi optimal atau terbaik untuk masalah yang dihadapi. Sebagai contoh, untuk masalah penjadwalan produksi, teknik-teknik "*searching*" digunakan untuk mencari jadwal produksi yang optimal yaitu jadwal yang dapat memberikan biaya yang paling kecil atau yang bisa meminimumkan waktu pembuatan barang.

Ada beberapa teknik "*searching*" yang dapat digunakan dalam bidang optimasi, di antaranya adalah *hill-climbing search, gradient search, simulated annealing,* dan *genetic algorithms. Hill-climbing* dan

gradient search adalah teknik *searching* tradisional yang menggunakan teori kalkulus dalam usahanya mencari titik maksimum atau titik minimum suatu fungsi dalam suatu "*search space*". Teknik ini cocok untuk digunakan jika turunan (derivatif) fungsi yang dioptimasi dapat ditentukan terlebih dahulu. Dengan kata lain, jika fungsi yang dihadapi tidak diketahui fungsi turunannya, teknik-teknik tersebut tidak dapat digunakan.

Simulated annealing menggunakan teknik "*searching*" yang hampir sama dengan teknik *gradient search.* Perbedaan utamanya terletak pada parameter pengukur kesuksesan hasil "*searching*". Dalam *simulated annealing,* ukuran yang digunakan adalah "temperatur". Penggunaan temperatur sebagai parameter "*searching*" ini adalah untuk meniru teknik yang dipakai para ahli metalurgi dalam membuat metal dengan sifat tertentu. Dalam membentuk metal, bahan-bahan dasarnya dipanaskan dan didinginkan berkali-kali sampai menghasilkan metal unggul yang mempunyai sifat tertentu, misalnya tidak mudah patah karena guncangan yang keras. Prinsip "*searching*" untuk *simulated annealing* juga seperti itu; jika temperatur "tinggi", perlu dibuat langkah yang besar (karena masih jauh dari titik optimum). Jika temperatur "rendah", yang diperlukan langkah kecil saja untuk menuju titik optimal. Pada awalnya, temperatur akan tinggi (karena masih jauh dari titik optimal). Namun selanjutnya, seiring dengan berjalannya waktu, temperatur akan turun perlahan-lahan sampai titik nol derajat, yang berarti titik optimum sudah tercapai. *Simulated annealing* telah banyak digunakan untuk menyelesaikan masalah optimasi jadwal dan rute transportasi di pabrik.

Berbeda dengan teknik-teknik di atas, *genetic algorithm* atau algoritma genetika melakukan "*searching*" dengan melakukan simulasi proses evolusi makhluk hidup. Prinsip utamanya yaitu meniru proses seleksi alam dan prinsip-prinsip ilmu genetika. Dalam seleksi alam, individu-individu bersaing untuk mempertahankan hidup dan

melakukan reproduksi. Individu-individu yang lebih "*fit*" akan mempunyai peluang untuk terus bertahan hidup (*survive*) dan melakukan reproduksi (menghasilkan keturunan). Sebaliknya, individu-individu yang kurang "fit" akan mati dan punah (prinsip ini dinamakan juga "*survival of the fittest*"). Kemudian dilanjutkan dengan proses penyilangan atau disebut pindah silang (*crossover*) serta mutasi. Kedua proses ini terjadi pada individu yang melakukan reproduksi sehingga menghasilkan individu baru *(child*). Proses seleksi dan reproduksi (pindah silang dan mutasi) ini berlangsung berulang kali, sampai mendapatkan individu terbaik

Penjelasan secara lebih terinci tentang algoritma genetika ini akan dibahas pada bab berikutnya. Pembahasan akan mencakup tentang konsep dasar, cara menentukan representasi kromosom, fungsi *fitness*, seleksi, penyilangan, dan mutasi. Pada bagian akhir bab tersebut juga diberikan satu contoh aplikasi algoritma genetika untuk ilustrasi.

Bab 2
Algoritma Genetika

Algoritma genetika adalah suatu teknik pencarian (*searching technique*) dan teknik optimasi yang cara kerjanya meniru proses evolusi dan perubahan struktur genetik pada makhluk hidup. Prinsip utama dari cara kerja algoritma genetika ini diilhami oleh proses seleksi alam dan prinsip-prinsip ilmu genetika. Dalam seleksi alam, individu-individu bersaing untuk mempertahankan hidup dan melakukan reproduksi. Individu-individu yang lebih "*fit*" akan mempunyai peluang untuk terus bertahan hidup (*survive*) dan melakukan reproduksi (menghasilkan keturunan). Sebaliknya, individu-individu yang kurang "fit" akan mati dan punah (prinsip ini dinamakan juga "*survival of the fittest*"). Selanjutnya, dalam proses seleksi alam ini, beberapa individu baru yang lebih "*fit*" dari kedua orang-tuanya akan "dilahirkan", melalui proses yang disebut penyilangan (*crossover*) dan mutasi. Kedua proses ini terjadi pada kromosom-kromosom individu yang melakukan reproduksi. Proses seleksi dan reproduksi (penyilangan dan mutasi) ini berlangsung berulang-kali, sampai individu yang paling "*fit*" dihasilkan. Individu yang paling *fit* inilah yang merupakan solusi dari masalah yang dihadapi.

Algoritma genetika pertama kali dikembangkan tahun 1970-an oleh John Holland (seorang profesor di Universitay of Michigan, Amerika). Tujuan yang ingin dicapai Holland saat itu adalah mengabstraksikan proses-proses evolusi yang terjadi di alam dan mendesain suatu *software* yang prinsip kerjanya meniru proses-proses evolusi. Hasilnya, algoritma genetika ternyata dapat menyelesaikan masalah-masalah yang tidak dapat diselesaikan dengan menggunakan

perhitungan matematika biasa. Algoritma genetika juga secara khusus dapat menyelesaikan masalah-masalah optimasi dengan baik. Suatu kejadian yang sangat penting dari penemuan ini adalah bahwa algoritma genetika hanya melibatkan operasi-operasi matematika yang sederhana.

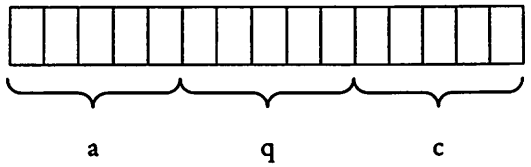
2.1 Kelahiran Algoritma Genetika

a. Penelitian Para Ahli Biologi

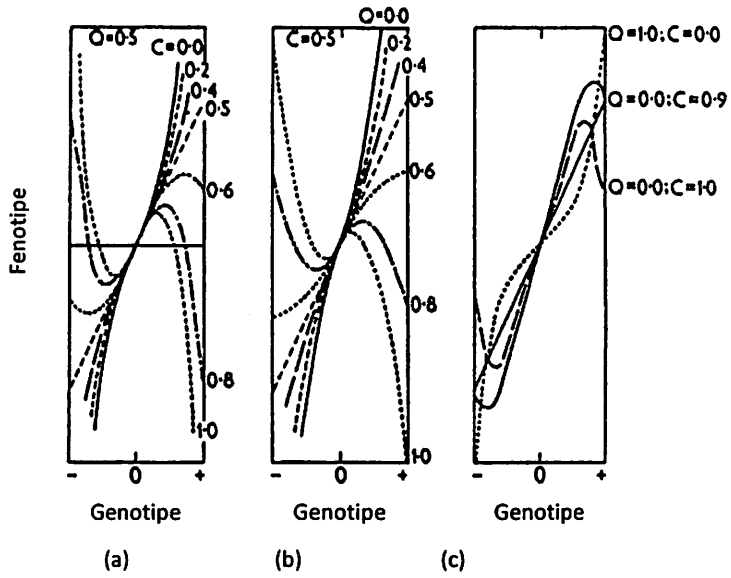
Sebelum ditemukannya algoritma genetika, para ahli biologi telah melakukan berbagai penelitian yang meniru proses evolusi dengan menggunakan komputer digital. Beberapa peneliti tersebut antara lain Baricelli (1957), Fraser (1960), Martin (1960), dan Cockerham (1960). Di antara penelitian-penelitian tersebut, adalah hasil penelitian Fraser yang paling mendekati proses algoritma genetika yang kita kenal sekarang ini. Fraser saat itu meneliti tentang epistasis, yaitu pengaruh genotipe (struktur gen) terhadap fenotipe (penampakan) suatu makhluk hidup. Dalam penelitiannya itu, Fraser memodelkannya dalam suatu fungsi matematis yaitu sebagai berikut:

$$\text{Fenotipe} = a + q.a\,|a| + c.a^3$$

Dari persamaan di atas, variabel a, q, dan c adalah variabel-variabel yang merepresentasikan tiga macam gen yang berbeda. Fraser meneliti pengaruh interaksi antara ketiga gen tersebut terhadap penampakan suatu individu (fenotipenya). Dalam simulasi di komputernya, Fraser menggunakan sejumlah string biner (diwakili oleh kode 0 dan 1) dengan panjang 15 digit. Lima bit pertama mewakili parameter a, lima bit berikutnya mewakili parameter q, dan lima bit terakhir mewakili parameter c. String biner tersebut dapat digambarkan sebagai berikut:

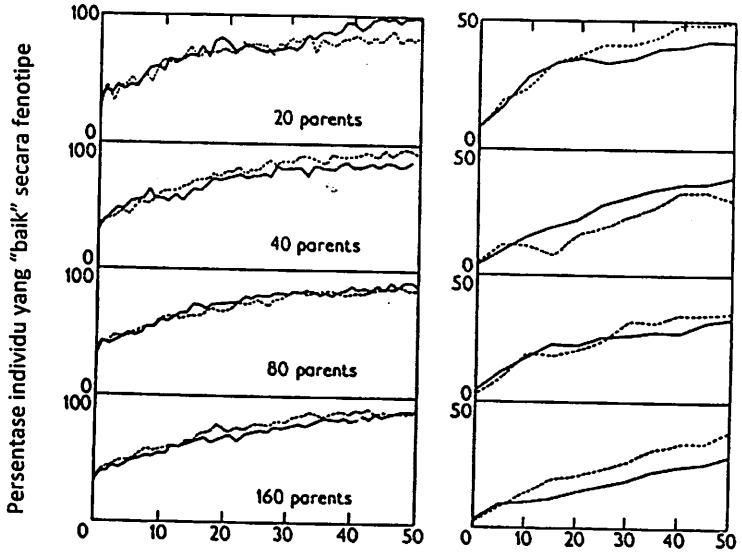


Dalam tahap seleksi untuk memunculkan kromosom induk, Fraser memanfaatkan string yang memiliki nilai fenotipe antara -1 dengan +1. Perbedaan nilai parameter a, q, dan c terhadap fenotipe telah memberikan pengaruh seperti yang dapat dilihat pada Gambar 2.1(a), 2.1(b), dan 2.1(c). Dari Gambar 2.1(a) dapat dilihat bahwa Fraser menggunakan variasi nilai a dengan nilai q dan c konstan yaitu sebesar 0,5 dan 0,0. Pada Gambar 2.1(b), nilai q menjadi 0,0 dan c menjadi 0,5. Sedangkan pada Gambar 2.1(c) diteliti pengaruh perbedaan antara variabel q dan c, dengan variabel a yang konstan.



Gambar 2.1 *Fraser's epistatic function* (sumber: Holland 1989)

Setelah mengetahui fenotipe dari masing-masing string, Fraser kemudian mengevolusikan string-string induk dan menghitung jumlah persentase individu yang fenotipenya cukup baik (antara -1 dengan +1). Hasil hitungan ini kemudian dituangkan dalam bentuk grafik seperti yang nampak pada Gambar 2.2. Dari Gambar 2.2 tersebut dapat kita lihat bahwa persentase individu yang mengalami perbaikan dari generasi ke generasi kian bertambah, bahkan mendekati 100%. Artinya, jumlah string yang pada generasi awal berfenotipe kurang baik, tetapi lama kelamaan semakin baik seiring bertambahnya jumlah generasi.



Gambar 2.2 Hasil simulasi Fraser (1962) menggunakan variasi jumlah populasi (sumber: Holland 1989)

John Holland, seorang profesor dari University of Michigan, terinspirasi oleh hasil penelitian Fraser tahun 1960 itu. Kala itu, ia pernah berujar:

"If evolution worked so well for organisms, why could it not work with computer programs"

Dari Gambar 2.2 dapat dilihat bahwa proses yang terjadi selama proses evolusi mirip dengan proses optimasi (maksimasi fungsi fenotipe). Pada awalnya, Fraser tidak berpikir bahwa riset yang dilakukannya itu adalah proses pencarian (*searching*) yang kelak menjadi pijakan bagi lahirnya suatu teknik pencarian dan optimasi yang sangat *robust* (tangguh) dan efisien, yaitu algoritma genetika (*genetic algorithms*).

John Holland kemudian membangun teori dan dasar-dasar algoritma genetika, dimulai dari Teori *Schemata*, operator-operator genetik, hingga membuat program komputernya. Selama kurang lebih tiga tahun (1962–1965), Holland aktif memberi kuliah tentang algoritma genetika di University of Michigan. Banyak di antara murid-muridnya yang kemudian menulis disertasi maupun makalah tentang algoritma genetika, di antaranya David Goldberg, Kenneth De Jong, Stephanie Forrest, dan Melanie Mitchell.

b. John Holland: Sang Pioner Algoritma Genetika

Pada awalnya, algoritma genetika yang diperkenalkan oleh John Holland belum mendapat sambutan yang cukup baik. Baru kemudian pada tahun 1980-an, algoritma genetika mulai mendapat perhatian, terutama di kalangan ilmuwan. Adalah David Goldberg, salah seorang kandidat doktor di USA dan juga merupakan murid John Holland, yang berhasil mengaplikasikan algoritma genetika dalam perancangan sistem perpipaan untuk distribusi gas alam. Goldberg memahami bahwa masalah perancangan pipa yang dikajinya itu sangat rumit. Oleh karena tertarik pada hasil penelitian Holland, Goldberg bertekad untuk menyelesaikan masalah perpipaan itu menggunakan algoritma genetika. Dari hasil riset yang dilakukannya, Goldberg berhasil membuktikan kalau algoritma genetika itu dapat bekerja dengan baik dalam

memecahkan masalah sistem perpipaan yang rumit. Hasil riset itu dituangkan dalam bukunya yang berjudul *Genetic Algorithm in Search, Optimization, and Machine Learning*, terbitan Addison-Wesley Publishing Company, tahun 1989. Buku itu kini menjadi buku pegangan utama (*handbook*) bagi para peneliti di dunia (termasuk para peneliti Indonesia) yang ingin mempelajari konsep-konsep dasar algoritma genetika.

Saat ini, algoritma genetika sudah mendapat sambutan yang sangat luar biasa dan telah banyak diaplikasikan di berbagai bidang. Beberapa di antaranya di bidang keteknikan (*engineering*), investasi, robotik, dan manajemen industri (seperti desain sistem produksi, tata letak fasilitas, penjadwalan produksi), dan otomatisasi industri. Untuk bidang agroindustri, algoritma genetika berpeluang besar untuk diaplikasikan pada desain sistem penyimpanan dan pengawetan hasil laut, sistem biorektor, dan prediksi permintaan produk agroindustri.

2.2 Konsep Dasar Algoritma Genetika

Tidak seperti teknik-teknik optimasi tradisional yang sudah ada sebelumnya, algoritma genetika bekerja tidak secara langsung pada parameter-parameter permasalahan yang ada. Sebaliknya, algoritma genetika mulai bekerja pada calon-calon solusi, yang dikodekan terlebih dahulu ke dalam bentuk kromosom. Proses untuk membuat kode atau membentuk struktur kromosom ini disebut sebagai proses pengkodean (*encoding*). Sebuah kromosom terdiri atas unit-unit terkecil yang disebut gen. Sebuah gen menggambarkan sebuah unit informasi yang terkandung dalam sebuah ruang pencarian (*search space*). Kumpulan dari gen-gen membentuk sebuah kromosom yang menggambarkan solusi masalah yang lengkap dan layak. Kromosom adalah simbol dalam bentuk string yang biasanya (tetapi tidak selalu) berbentuk bilangan biner, yaitu kombinasi angka 0 dan 1. Angka 0 dan 1 ini merupakan

sebuah gen. Kumpulan dari kromosom-kromosom membentuk sebuah komunitas yang dinamakan populasi.

Kunci dari kekuatan algoritma genetika terletak pada kromosom itu sendiri di mana kromosom tidak mengetahui makna yang terkandung di dalamnya. Sehingga untuk mencari solusi yang paling baik atau paling optimal, digunakanlah fungsi *fitness*. Fungsi *fitness* berfungsi untuk mengukur seberapa bagus atau seberapa "*fit*" sebuah kromosom. Proses pengkodean kromosom dapat dilakukan menggunakan bilangan biner.

2.3 Representasi Kromosom

Untuk dapat mengaplikasikan algoritma genetika, langkah pertama yang harus dilakukan adalah membuat pengkodean (*encoding*) calon solusi ke dalam suatu bentuk representasi kromosom. Representasi kromosom yang pertama kali diperkenalkan oleh Holland (1975) adalah representasi *string biner*. Dalam representasi ini, sebuah kromosom terdiri atas beberapa elemen yang disimbolkan dengan angka nol (0) dan satu (1). Setiap untaian elemen memiliki arti khusus yang menunjukkan nilai *fitness* kromosom yang bersangkutan.

Bentuk representasi biner yang diperkenalkan oleh Holland (1975), kurang cocok dipakai dalam memecahkan masalah kombinatorial (Gen dan Cheng 1997), seperti masalah *Travelling Salesman Problem* (TSP) dan masalah penjadwalan *flow-shop*. Akhir-akhir ini, beberapa peneliti telah memperkenalkan beberapa bentuk representasi baru (*non-biner*) sesuai dengan masalah yang akan dipecahkan. Contoh representasi *non-biner* ini adalah representasi bilangan integer dan representasi matrix. Penggunaan representasi ini akan dibahas pada bab selanjutnya.

2.4 Fungsi *Fitness* dan Fungsi Tujuan

Algoritma genetika bekerja dengan mengukur seberapa baik sebuah kromosom dapat menyelesaikan suatu masalah. Pengukuran dilakukan dengan menggunakan fungsi *fitness*, yaitu fungsi tujuan dari masalah yang hendak diselesaikan. Fungsi *fitness* hanya menggunakan satu set parameter masalah dan mengembalikan ukuran kegunaannya (efisiensi, biaya, dan sebagainya). Semakin besar nilai *fitness*, semakin bugar pula kromosom dalam populasi sehingga semakin besar kemungkinan kromosom tersebut untuk tetap *survive* pada generasi berikutnya.

Suatu fungsi *fitness* dapat sama atau merupakan hasil modifikasi terhadap fungsi tujuan masalah yang hendak diselesaikan. Jika masalahnya adalah masalah maksimasi, fungsi *fitness*-nya sama atau berbanding lurus dengan fungsi tujuan. Namun jika masalahnya adalah masalah minimasi, fungsi *fitness*-nya berbanding terbalik dengan fungsi tujuan. Secara umum, algoritma genetika mengoperasikan kromosom dengan *fitness* positif dan mencari untuk memaksimumkan *fitness* ini. Proses minimasi dapat dilakukan dengan mudah, dengan cara membalik fungsi maksimasi.

2.5 Seleksi Kromosom

Seleksi merupakan salah satu operasi untuk memastikan bahwa jumlah perwakilan dari sebuah kromosom yang diterima pada generasi selanjutnya akan bergantung pada nilai *fitness*-nya yang dibandingkan dengan nilai *fitness* rata-rata dari populasi yang ada. Kromosom-kromosom yang telah dievaluasi dengan menggunakan fungsi *fitness* akan diseleksi untuk dijadikan induk. Kromosom-kromosom yang memiliki nilai *fitness* yang sangat baik akan memiliki peluang yang lebih besar untuk terpilih menjadi induk dan tetap bertahan pada generasi

berikutnya, sedangkan kromosom-kromosom yang lebih buruk akan tergantikan oleh kromosom baru.

Pada model seleksi alam, kromosom yang memiliki nilai *fitness* lebih baik akan memiliki peluang bertahan hidup (*survival of the fittest*) yang lebih baik pada generasi berikutnya. Sebaliknya, kromosom yang memiliki nilai *fitness* buruk akan tergantikan oleh kromosom baru yang lebih baik. Proses seleksi dalam algoritma genetika juga meniru prinsip seleksi alam dalam cara kerjanya. Salah satu prinsip umum seleksi adalah peluang masing-masing kromosom untuk terpilih sebanding dengan nilai *fitness*-nya. Tipe seleksi ini disebut *fitness proportional selection*. Jadi, jika kromosom A dua kali nilainya dari kromosom B, kromosom A seharusnya memiliki dua kali kesempatan untuk bereproduksi.

Salah satu teknik seleksi dalam algoritma genetika adalah teknik seleksi cakram rolet (*roulette wheel selection*) yang dikenalkan oleh Goldberg (1989). Teknik seleksi ini diilustrasikan sebagai teknik pemutaran cakram rolet. Setiap kromosom dalam populasi menempati suatu slot pada cakram rolet. Besarnya ukuran slot adalah sama dengan rasio antara nilai *fitness* suatu kromosom dengan total nilai *fitness* semua kromosom. Untuk menghasilkan sejumlah populasi, rolet tersebut diputar sebanyak ukuran populasi yang ada.

2.6 Operator-operator Algoritma Genetika

Tujuan utama yang ingin dicapai dalam menggunakan algoritma genetika adalah agar sekumpulan kromosom (calon solusi) yang dibangkitkan secara acak pada populasi awal dapat berkembang biak dengan sendirinya (melalui beberapa generasi) hingga konvergen memberikan suatu nilai *fitness* yang terbaik (nilai optimal). Kromosom yang terbentuk pada generasi baru disebut kromosom anak (*offspring*).

Sebuah kromosom anak dapat terbentuk melalui dua proses utama yaitu dengan menggabungkan elemen-elemen antara dua kromosom induk (*parents*) menggunakan operator penyilangan, atau dengan memodifikasi sebuah kromosom induk menggunakan operator mutasi. Dalam satu generasi, kedua proses di atas dapat terjadi secara berurutan (*sequential*) maupun paralel. Maksud secara berurutan adalah proses penyilangan terjadi lebih dahulu pada dua kromosom induk, lalu dilanjutkan dengan proses mutasi pada kedua kromosom anak yang baru terbentuk. Proses semacam ini disebut *mutation embedded within crossover*. Sedangkan maksud secara paralel adalah proses penyilangan dan mutasi terjadi secara terpisah pada kromosom-kromosom induk saja, tidak pada kromosom anak yang baru terbentuk.

2.6.1 Penyilangan

Penyilangan (*crossover*) adalah operator utama atau operator primer dalam algoritma genetika. Operator ini bekerja pada sepasang kromosom induk untuk menghasilkan dua kromosom anak dengan cara menukarkan beberapa elemen (gen) yang dimiliki masing-masing kromosom induk. Tingkat penyilangan atau peluang penyilangan (dinotasikan sebagai *Pc*) adalah rasio antara jumlah kromosom yang diharapkan mengalami penyilangan dalam setiap generasi dengan jumlah kromosom total dalam populasi. Oleh karena penyilangan adalah operator primer, nilai *Pc* yang digunakan biasanya cukup tinggi (0,6–1). Tingkat penyilangan yang tinggi menyebabkan semakin besarnya kemungkinan algoritma genetika mengeksplorasi ruang pencarian sekaligus mempercepat ditemukannya solusi optimum. Akan tetapi, apabila tingkat penyilangan terlalu tinggi, hal ini sama artinya dengan membuang-buang waktu mencari solusi pada daerah yang mungkin saja kurang menjanjikan (*unpromising region*). Penentuan nilai *Pc* yang tepat sangat bergantung pada permasalahan yang dihadapi.

Holland (1975) telah memperkenalkan operator penyilangan yang disebut penyilangan satu-titik (*one-point crossover*). Penyilangan satu-titik ini cocok digunakan untuk kromosom dengan representasi biner (0 dan 1). Pada beberapa kasus, seperti masalah TSP (*Travelling Salesman Problems*), penyilangan satu-titik ini tidak cocok digunakan karena dapat menghasilkan kromosom ilegal. Atas dasar itulah, beberapa peneliti telah mengusulkan teknik penyilangan yang baru, seperti *partially-mapped crossover* (PMX), *order crossover* (OX), dan *cycle-croossover* (CX), *position-based crossover*, *order-based crossover*, dan *heuristic crossover* (Gen dan Cheng 1997). Teknik PMX pertama kali diperkenalkan oleh Goldberg dan Lingle (1985). Teknik PMX dapat dipandang sebagai perbaikan dari teknik penyilangan dua-titik (*two-point crossover*) yaitu adanya metode perbaikan (*repairing procedure*) untuk mencegah munculnya kromosom yang ilegal.

2.6.2 Mutasi

Mutasi (*mutation*) adalah operator sekunder atau operator pendukung dalam algoritma genetika yang berperan mengubah struktur kromosom secara spontan. Perubahan spontan ini menyebabkan terbentuknya suatu **mutan**, yaitu suatu kromosom baru yang secara genetik berbeda dari kromosom sebelumnya. Operator mutasi bekerja pada satu kromosom, tidak pada sepasang kromosom seperti halnya yang dilakukan operator penyilangan. Dalam pencarian solusi optimum, mutasi sangat diperlukan yaitu untuk: (1) mengembalikan gen-gen yang hilang pada generasi-generasi sebelumnya , dan (2) memunculkan gen-gen baru yang belum pernah muncul pada generasi-generasi sebelumnya (Gen dan Cheng 1997).

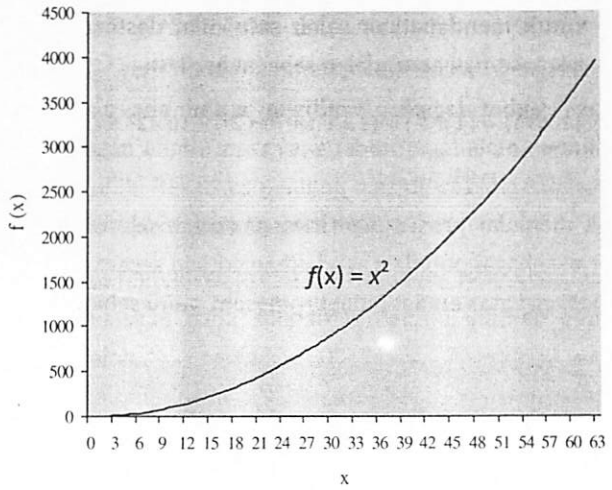
Tingkat mutasi atau peluang mutasi (dinotasikan sebagai *Pm*) adalah rasio antara jumlah gen yang diharapkan mengalami mutasi pada setiap generasi dengan jumlah gen total dalam populasi. Oleh karena mutasi adalah operator sekunder, nilai *Pm* yang digunakan untuk

running program biasanya cukup rendah (0,001–0,2). Jika tingkat mutasi terlalu rendah, semakin kecil pula kemungkinan memunculkan gen-gen baru. Padahal, gen yang baru sebenarnya sangat diperlukan dalam menunjang keberhasilan memperoleh solusi optimum. Sebaliknya, jika tingkat mutasi terlalu tinggi, akan banyak sekali mutan yang muncul. Akibatnya, banyak karakteristik kromosom induk yang kemungkinan hilang pada generasi berikutnya sehingga algoritma genetika akan kehilangan kemampuan untuk mengingat atau belajar dari proses pencarian sebelumnya (Gen dan Cheng 1997).

Gen dan Cheng (1997) menyebutkan bahwa dalam beberapa tahun belakangan ini para peneliti telah memperkenalkan beberapa jenis operator mutasi. Beberapa operator mutasi untuk masalah yang menggunakan *permutation representation* atau *order representation* adalah mutasi inversi, mutasi insersi, *diplacement mutation*, dan *reciprocal exchange mutation.*

2.7 Contoh Ilustrasi Algoritma Genetika Sederhana (AGS)

Pada bagian ini akan dibahas secara rinci tentang bagaimana algoritma genetika bekerja dalam proses pencarian suatu nilai optimum fungsi matematis. Untuk itu, penulis melakukan eksperimen menggunakan bahasa pemrograman Pascal untuk mencari nilai maksimum dari fungsi matematis $f(x) = x^2$ dengan wilayah pencariannya adalah $0 \leq x \leq 63$. Hasil eksperimen tersebut, penulis sajikan dalam bentuk grafik seperti yang nampak pada Gambar 2.3.



Gambar 2.3 Grafik fungsi untuk Algoritma Genetika Sederhana

Untuk menyelesaikan masalah di atas dengan menggunakan algoritma genetika, langkah-langkah yang penulis lakukan adalah sebagai berikut.

Representasi kromosom

Kromosom yang terdiri atas 6 digit biner dibuat untuk mengkodekan wilayah pencarian Kromosom-kromosom tersebut adalah:

Kromosom	Nilai desimal
000000	0
000001	1
...	...
110011	51
011011	27

Cara untuk mendapatkan salah satu nilai desimal dari contoh kromosom-kromosom di atas adalah sebagai berikut:

$$\text{Nilai desimal } (110011) = (1 \times 2^5) + (1 \times 2^4) + (0 \times 2^3) + (0 \times 2^2 + (1 \times 2^1) + (1 \times 2^0)$$

$$= 51$$

Untuk memulai proses algoritma genetika, sejumlah kromosom yang berperan sebagai populasi awal dibangkitkan **secara acak**. Misalkan populasi ini terdiri atas empat buah kromosom, yaitu sebagai berikut.

No.	Kromosom pada populasi awal	Nilai desimal
1.	110011	51
2.	010111	23
3.	101100	44
4.	011011	27

Nilai *fitness* keempat kromosom di atas dapat diketahui dengan mensubstitusikan nilai desimal masing-masing kromosom ke dalam fungsi *fitness*. Hasil penghitungan ini dapat dilihat pada Tabel 2.1.

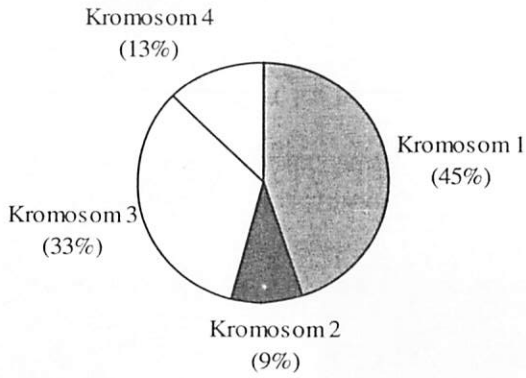
Tabel 2.1 Hasil penghitungan untuk contoh Algoritma Genetika Sederhana

No.	Kromosom pada populasi awal	Nilai desimal (x)	Nilai fitness $f(x)=x^2$
1.	110011	51	2601
2.	010111	23	529
3.	101100	44	1936
4.	011011	27	729
Total			5795
Rata-rata			1448,75
Maksimum			2601

Seleksi

Sebelum reproduksi dimulai, dilakukan seleksi terhadap kromosom yang ada untuk dijadikan sebagai induk. Secara alami, kromosom dengan nilai *fitness* yang lebih tinggi akan memiliki peluang lebih besar terpilih. Besarnya peluang masing-masing kromosom dalam contoh ini dapat dilihat pada Gambar 2.4.

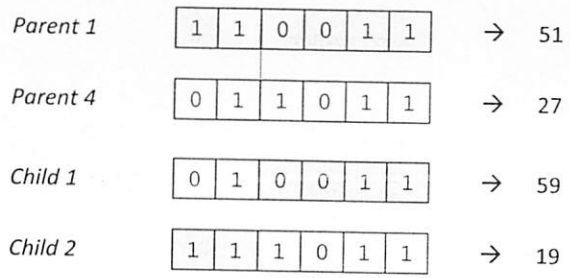
Salah satu teknik seleksi adalah *Roulette Wheel Selection*, seperti yang telah dijelaskan pada subbab terdahulu. Jumlah populasi yang digunakan adalah 4 kromosom. Dengan demikian, agar jumlah populasinya konstan, roda rolet diputar sebanyak 4 kali. Setiap pemutaran roda rolet akan dihasilkan satu buah kromosom induk. Setelah roda rolet diputar empat kali, misalkan secara berurutan diperoleh kromosom 1, kromosom 3, kromosom 2, dan kromosom 4. Kromosom-kromosom inilah yang akan menjadi kromosom induk, untuk kemudian akan mengalami penyilangan dan atau juga mutasi.



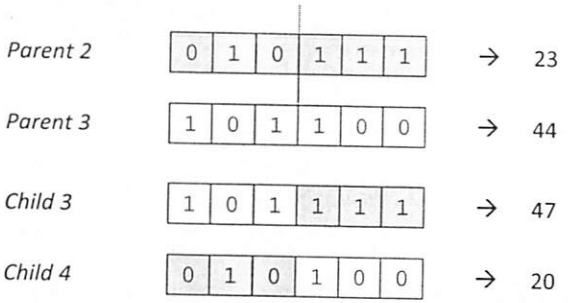
Gambar 2.4 Proporsi peluang terpilihnya kromosom dalam tahap *Roulette Wheel Selection*

Penyilangan (*Crossover*)

Ilustrasi proses penyilangan kromosom 1 (*Parent 1)* dan kromosom 4 (*Parent 4*) dengan titik potong pada posisi 2 yang ditentukan secara *random* (acak) adalah sebagai berikut:



Ilustrasi proses penyilangan kromosom 2 (*Parent 2)* dan kromosom 3 (*Parent 3*) dengan titik potong pada posisi 3 yang ditentukan secara *random* (acak) adalah sebagai berikut:



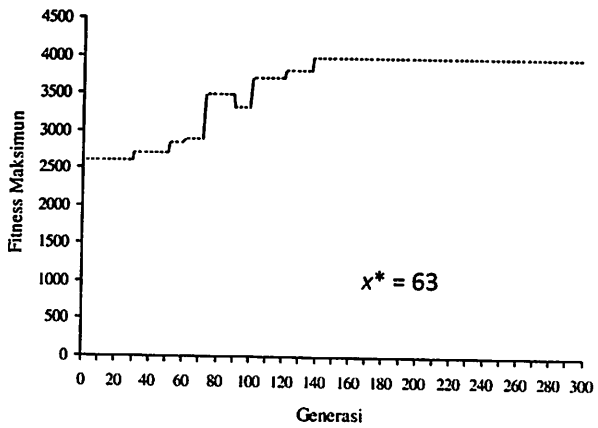
Berdasarkan penjelasan di atas, dapat dilihat bahwa proses penyilangan akan menghasilkan keturunan yang lebih *fit* (yang memiliki nilai *fitness* lebih tinggi). Populasi yang dihasilkan disajikan pada Tabel 2.2.

Tabel 2.2 Populasi baru pada contoh Algoritma Genetika Sederhana

	Kromosom anak	Nilai desimal (x)	Nilai *fitness* $f(x)=x^2$
1	111011	59	3481
2	010011	19	361
3	101111	47	2209
4	010100	20	400
		Total	6451
		Rata-rata	1612,75
		Maksimum	3481

Dari Tabel 2.2 dapat dilihat bahwa nilai total dan nilai *fitness* rata-rata dari populasi baru lebih tinggi dari populasi awal. Hal ini menandakan bahwa proses pencarian menggunakan algoritma genetika mampu mendapatkan nilai x yang lebih baik dari yang sebelumnya. Di mana kromosom dengan nilai x yang baik merupakan calon optimal permasalahan yang dihadapi. Untuk mempertahankan kromosom yang memiliki nilai yang optimal maka diperlukan proses elitisme serta penambahan generasi sehingga menghasilkan solusi optimal seperti pada Gambar 2.5.

Dari Gambar 2.5 di atas dapat kita lihat bahwa nilai *fitness* dari generasi ke generasi kian membaik (meningkat). Peningkatan ini dimulai pada generasi ke-30 hingga generasi ke-40. Setelah itu, nilai *fitness* akan konvergen hingga generasi ke-300. Saat konvergen inilah, dicapai nilai x optimal (x^*) yaitu 63. Struktur kromosom untuk x ini adalah 111111 dengan nilai *fitness* 3969. Nilai ini adalah nilai *fitness* terbaik (paling optimum) yang pernah ditemukan oleh algoritma genetika.



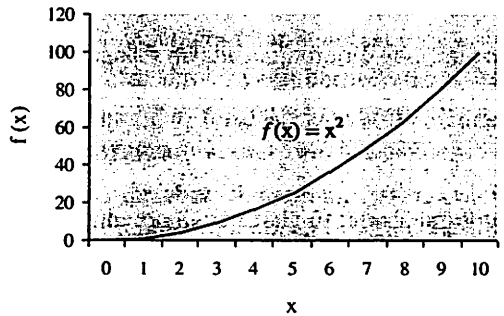
Gambar 2.5 Grafik nilai *fitness* maksimum untuk fungsi $f(x) = x^2$

Bab 3
Contoh Kasus Masalah Optimasi

Bab 3 akan membahas beragam pemecahan masalah menggunakan algoritma genetika. Masalah yang dipecahkan meliputi masalah optimasi fungsi sederhana $f(x)=x^2$, masalah *Word-Matching Problem* (WMP), dan masalah optimasi fungsi sulit (*hard optimization problems*). Berikut adalah pembahasannya.

3.1 Kasus 1: Optimasi Fungsi Sederhana $f(x) = x^2$

Seperti contoh pada Bab 2, yaitu kasus Algoritma Genetika Sederhana (AGS), pada bagian ini juga akan dibahas secara rinci tentang bagaimana algoritma genetika bekerja dalam mencari nilai optimum suatu fungsi matematis. Misalkan akan dicari nilai maksimum dari fungsi *fitness* $f(x) = x^2$ dengan wilayah pencariannya adalah $0 \leq x \leq 10$. Grafik fungsi ini dapat dilihat pada Gambar 3.1 di bawah ini.



Gambar 3.1 Grafik fungsi pencarian untuk kasus 1

Untuk menyelesaikan masalah tersebut, langkah-langkah yang dilakukan adalah sebagai berikut:

Representasi kromosom

Langkah pertama yang dilakukan adalah mengkodekan wilayah pencarian ke dalam kromosom biner. Panjang kromosom bergantung pada tingkat ketelitian yang diinginkan. Misalkan variabel pencariannya adalah x di mana x terletak $[a, b]$ dan tingkat ketelitiannya adalah empat angka di belakang koma. Tingkat ketelitian ini menunjukkan bahwa daerah pencarian yang kontinu itu harus dipartisi oleh sedikitnya $(b - a) \times 10^4$. Panjang kromosom yang diperlukan (disimbolkan oleh *Lchrom*) dihitung dengan menggunakan rumus:

$$2^{Lchrom-1} < (b-a) \times 10^4 \leq 2^{Lchrom} - 1$$

Oleh karena kromosom yang digunakan berbentuk biner, deret biner ini perlu dikonversi ke dalam bentuk desimal. Pengonversian dilakukan dengan menggunakan rumus sebagai berikut:

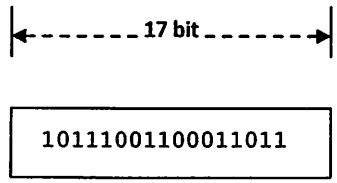
$$x = a + desimal(string) \times \frac{b-a}{2^{Lchrom} - 1}$$

di mana *desimal(string)* merepresentasikan nilai desimal dari variabel x.

Panjang kromosom yang dibutuhkan adalah sebagai berikut:

$$(10-0) \times 10^4 = 10^5$$
$$2^{16} < 10^5 \leq 2^{17} \quad \rightarrow Lchrom = 17$$

Jadi, untuk mencapai tingkat ketelitian empat angka di belakang koma, panjang kromosom yang dibutuhkan adalah 17 bit. Misalkan kromosom v_j direpresentasikan sebagai berikut:



Nilai desimal untuk kromosom v_j di atas adalah 95003, dan nilai variabel x dihitung sebagai berikut:

$$x = 0 + 95003 \times \frac{10-0}{2^{17} - 1} = 7{,}2482$$

Nilai *fitness* untuk kromosom v_j dengan nilai x sebesar 7,2482 adalah:

$$fitness \quad v_j = f(7{,}2482) = (7{,}2482)^2 = 52{,}5365$$

Hasil Eksperimen

Dalam eksperimen, nilai-nilai parameter algoritma genetika yang digunakan adalah sebagai berikut: jumlah populasi (*PopSize*)=20, generasi maksimum (*MaxGen*)=600, peluang penyilangan (*Pc*)=0.85, dan peluang mutasi (*Pm*)=0,10. Hasil eksperimen tersebut disajikan dalam 2 buah gambar yaitu Gambar 3.2 dan Gambar 3.3. Penjelasan masing-masing gambar seperti pada gambar 3.2.

Pada Gambar 3.2 dapat kita lihat generasi awal yang terdiri atas 20 kromosom dengan nilai *fitness* yang berbeda-beda. Nilai *fitness* ini diperoleh secara langsung dari fungsi $f(x)=x^2$. Nilai *fitness* maksimum yang muncul adalah 60,9848 dan nilai *fitness* minimum adalah 0,0022. Kromosom-kromosom tersebut kemudian akan diseleksi untuk dijadikan kromosom induk. Kromosom-kromosom yang terpilih kemudian akan dikembangbiakkan (disilangkan atau dimutasikan) untuk membentuk keturunan-keturunan pada generasi berikutnya.

Tentu saja kromosom yang memiliki nilai *fitness* besar memiliki peluang untuk terpilih menjadi kromosom induk.

Berikutnya, pada Gambar 3.3 dapat kita lihat grafik perubahan nilai *fitness* (rata-rata dan maksimum) hingga mencapai 600 generasi. Nilai maksimum yang didapat ialah dengan nilai x = 10, dan f(x) = 100. Untuk nilai *fitness* rata-rata, terjadi peningkatan yang cukup signifikan hingga generasi ke-70, tetapi kemudian antar generasi ke-70 dan generasi ke-100 terjadi penurunan dikarenakan tidak terdapatnya proses elitisme untuk mempertahankan kromosom yang baik. Secara umum proses pencarian yang dilakukan algoritma genetika sudah optimal dengan berhasil menemukan nilai x yang maksimum.

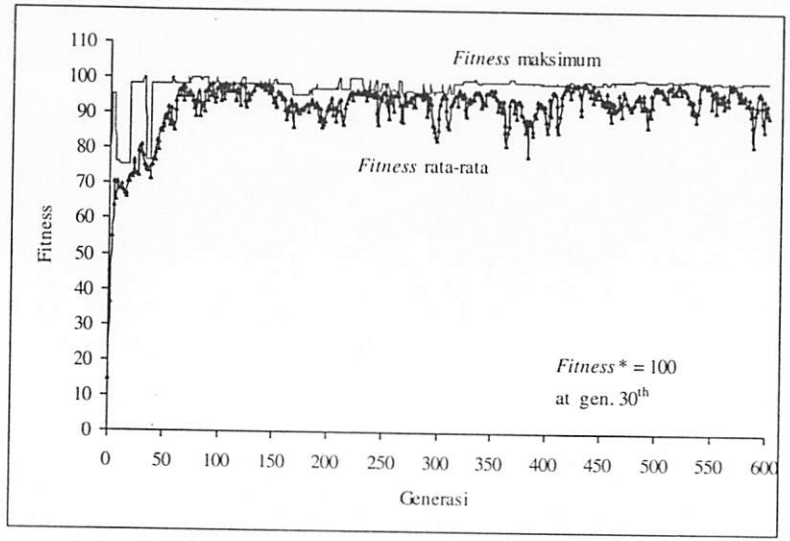

```
================
 INITIAL REPORT
================

GAs parameters
==============
Pop. Size            = 20
Chrom. Length        = 17
Max. Generation      = 600
Crossover probability    = 0.8500
Mutation probability     = 0.1000

INITIAL GENERATION STASTISTICS
==============================
       Chromosome          Decimal          x     Fitness = x^2
 1) 00001010000110001          5169     0.3944         0.1555
 2) 00100011101101011         18283     1.3949         1.9457
 3) 01010011011010000         42704     3.2581        10.6151
 4) 00111000010010110         28822     2.1990         4.8354
 5) 10111001100011011         95003     7.2482        52.5365
 6) 01000010100011000         34072     2.5995         6.7574
 7) 01001001001111010         37498     2.8609         8.1847
 8) 11000111111010101        102357     7.8093        60.9848
 9) 01011101111111010         48122     3.6714        13.4795
10) 01111000100000111         61703     4.7076        22.1615
11) 11000001000011010         98842     7.5411        56.8682
12) 00100000010111011         16571     1.2643         1.5984
13) 01011110011101111         48367     3.6901        13.6171
14) 00110001100001011         25355     1.9344         3.7421
15) 00000001001100111           615     0.0469         0.0022
16) 01000011010110001         34481     2.6307         6.9206
17) 00010010001010100          9300     0.7095         0.5034
18) 00011110010110110         15542     1.1858         1.4060
19) 01010000001001000         41032     3.1305         9.8001
20) 01100101010001110         51854     3.9562        15.6513

Sum of Fitness =   291.7660
Max. Fitness   =    60.9848
Min. Fitness   =     0.0022
Avg. Fitness   =    14.5883
```

Gambar 3.2 Statistik generasi awal untuk kasus 1



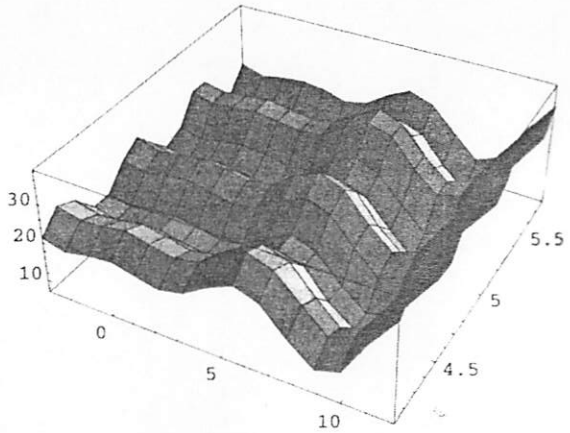
Gambar 3.3 Grafik nilai *fitness* untuk kasus 1

3.2 Kasus 2: Optimasi Fungsi Sulit-1

Pada kasus ketiga ini, penulis melakukan eksperimen menggunakan fungsi sulit-berkendala, yang diterapkan dalam bahasa pemrograman *Pascal*. Fungsi sulit ini telah digunakan juga oleh Michalewicz (1996) serta Gen dan Cheng (1997) untuk membuktikan ketangguhan algoritma genetika. Hasil penelitian mereka menunjukkan bahwa algoritma genetika cukup tangguh (*robust*) untuk optimasi fungsi yang sulit, yang tidak bisa diselesaikan dengan teknik-teknik optimasi konvensional. Adapun fungsi sulit yang digunakan oleh penulis adalah seperti di bawah ini.

$$Max\ f(x_1, x_2) = 21.5 + x_1\ Sin(4\pi x_1) + x_2\ Sin(20\pi x_2)$$
$$-3{,}0 \le x_1 \le 12{,}1$$
$$4{,}1 \le x_2 \le 5{,}8$$

Berdasarkan eksperimen yang telah dilakukan oleh Gen dan Cheng (1997), keduanya memperlihatkan hasil temuannya dalam bentuk grafik tiga dimensi, yaitu seperti pada Gambar 3.4.



Gambar 3.4 Grafik fungsi pencarian untuk kasus 2 (sumber: Gen dan Cheng 1997)

Adapun langkah-langkah penyelesaian terhadap fungsi sulit di atas, adalah sebagai berikut:

Representasi Kromosom

Langkah pertama yang dilakukan adalah mengkodekan variabel-variabel keputusan menjadi kromosom yang berbentuk biner. Panjang setiap kromosom bergantung pada ketelitian yang dikehendaki (roses penentuan panjang kromosom ini telah dibahas pada Kasus 1).

Pada Kasus 2 ini, misalkan variabel pencariannya adalah x_j yang nilainya berada dalam rentang $[a_j, b_j]$. Tingkat ketelitian yang dikehendaki adalah lima angka di belakang koma. Untuk memenuhi tingkat ketelitian ini, daerah pencarian harus dipartisi menjadi sedikitnya $(b_j - a_j).10^5$ bagian. Panjang kromosom yang dikehendaki

(disimbolkan m_j) untuk sebuah variabel dapat dihitung dengan menggunakan rumus sebagai berikut:

$$2^{m_j-1} < (b_j - a_j) \times 10^5 \leq 2^{m_j} - 1$$

Pengonversian bilangan biner ke sebuah bilangan desimal untuk variabel x_j adalah sebagai berikut:

$$x_j = a_j + decimal(substring_j) \times \frac{b_j - a_j}{2^{m_j} - 1}$$

di mana $decimal(substring_j)$ mewakili nilai desimal dari $substring_j$.

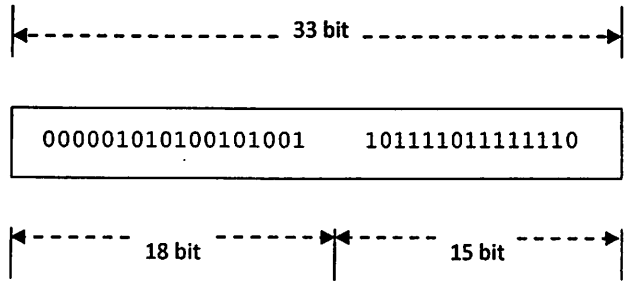
Dengan tingkat ketelitian lima angka di belakang koma, panjang kromosom yang diperlukan untuk variabel x_1 dan x_2 adalah:

$$(12{,}1-(-3{,}0)) \times 10^4 = 151000$$
$$2^{17} < 151000 \leq 2^{18} \quad \rightarrow m_1 = 18$$

$$(5{,}8-4{,}1) \times 10^4 = 17000$$
$$2^{14} < 17000 \leq 2^{15} \quad \rightarrow m_2 = 15$$

$$m = m_1 + m_2 = 18+15 = 33$$

Total panjang dari sebuah kromosom adalah 33 bit yang digambarkan sebagai berikut:



Nilai yang sesuai untuk variabel x_1 dan x_2 diberikan di bawah ini:

	Kromosom	Nilai desimal
x1	000001010100101001	5417
x2	101111011111110	24318

$$x_1 = -3.0 + 5417 \times \frac{12{,}1-(-3{,}0)}{2^{18}-1} = -2{,}687969$$

$$x_2 = 4{,}1 + 24318 \times \frac{5{,}8-4{,}1}{2^{15}-1} = 5{,}361653$$

Populasi Awal

Populasi awal yang dibangkitkan secara acak adalah sebagai berikut:

v_1 = [000001101111000000010011110010100]
v_2 = [101101010100110100111111101001010]
v_3 = [001001100111110110001111010100000]
v_4 = [101101000100011000100101101111101]
v_5 = [100011101011001110111101011010101]
v_6 = [010001000111100011110011100100101]

v_7 = [11000111010001100110110100011110]
v_8 = [00101101011010010011011000101001 10]
v_9 = [11110110010000000010110001011011 1]
v_{10} = [00100110000000100110101001001 1000]
v_{11} = [10110010100001110110111101000011 0]
v_{12} = [01110110000010001100010000101001 1]
v_{13} = [10010110100010101101101100001100 0]
v_{14} = [10101010100010111110111010110010 0]
v_{15} = [10001100011010100111001111001011 0]
v_{16} = [00010111110000111111000100110011 00]
v_{17} = [00111110001110101100011111011000 1]
v_{18} = [01100000000000101110000011000000 00]
v_{19} = [01001000001111101111100011111010 0]
v_{20} = [11011111100101010011110110110011 0]

Nilai variabel x1 dan x2 yang bersesuaian adalah:

v_1 = [x1; x2] = [-2,590794 ; 4,625663]
v_2 = [x1; x2] = [7,693954 ; 5,790609]
v_3 = [x1; x2] = [-0,729669 ; 4,506751]
v_4 = [x1; x2] = [7,633357 ; 5,102609]
v_5 = [x1; x2] = [5,417172 ; 5,731413]
v_6 = [x1; x2] = [1,038775 ; 5,469930]
v_7 = [x1; x2] = [8,754122 ; 5,298566]
v_8 = [x1; x2] = [-0,321500 ; 5,543031]
v_9 = [x1; x2] = [11,524958 ; 5,278280]
v_{10} = [x1; x2] = [-0,758067 ; 5,223545]
v_{11} = [x1; x2] = [7,530422 ; 5,355428]
v_{12} = [x1; x2] = [3,962199 ; 4,210559]

v_{13} = [x1; x2] = [5,879659 ; 4,818455]
v_{14} = [x1; x2] = [7,059582 ; 5,340382]
v_{15} = [x1; x2] = [5,282325 ; 5,475793]
v_{16} = [x1; x2] = [-1,598252 ; 5,013736]
v_{17} = [x1; x2] = [0,670582 ; 4,308408]
v_{18} = [x1; x2] = [2,663155 ; 4,989871]
v_{19} = [x1; x2] = [1,261349 ; 5,613486]
v_{20} = [x1; x2] = [10,187897 ; 5,738936]

Evaluasi Kromosom

Tahap evaluasi atau tahap penghitungan nilai *fitness* sebuah kromosom adalah sebagai berikut.

1. Konversi struktur kromosom yang berdigit biner ke dalam suatu bilangan desimal, yaitu bilangan real $x^k = (x_1^k, x_2^k), \quad k = 1,2,...pop\_size$
2. Tentukan nilai fungsi tujuan $f(x^k)$.
3. Tentukan nilai *fitness*. Untuk masalah maksimasi, fungsi *fitness* sama fungsi tujuan yaitu $Eval(v_k) = f(x^k), \quad k = 1,2,...pop\_size.$

Nilai *fitness* setiap kromosom-kromosom di atas adalah sebagai berikut:

$Eval(v_1)$ = f (-2,590794 : 4,625663) = 28,476765
$Eval(v_2)$ = f (7,693954 : 5,790609) = 23,260013
$Eval(v_3)$ = f (-0,729669 : 4,506751) = 23,539172
$Eval(v_4)$ = f (7,633357 : 5,102609) = 29,924133
$Eval(v_5)$ = f (5,417172 : 5,731413) = 22,098307
$Eval(v_6)$ = f (1,038775 : 5,469930) = 16,791618
$Eval(v_7)$ = f (8,754122 : 5,298566) = 20,569913
$Eval(v_8)$ = f (-0,321500 : 5,543031) = 23,598764

$Eval(v_9)$	= f (	11,524958	:	5,278280) =	19,889001
$Eval(v_{10})$	= f (	-0,758067	:	5,223545) =	26,625019
$Eval(v_{11})$	= f (	7,530422	:	5,355428) =	22,518002
$Eval(v_{12})$	= f (	3,962199	:	4,210559) =	22,280935
$Eval(v_{13})$	= f (	5,879659	:	4,818455) =	20,047091
$Eval(v_{14})$	= f (	7,059582	:	5,340382) =	29,339872
$Eval(v_{15})$	= f (	5,282325	:	5,475793) =	13,943805
$Eval(v_{16})$	= f (	-1,598252	:	5,013736) =	26,818499
$Eval(v_{17})$	= f (	0,670582	:	4,308408) =	24,235199
$Eval(v_{18})$	= f (	2,663155	:	4,989871) =	20,897234
$Eval(v_{19})$	= f (	1,261349	:	5,613486) =	25,528217
$Eval(v_{20})$	= f (	10,187897	:	5,738936) =	32,343953

Dengan membandingkan nilai-nilai di atas, terlihat dengan jelas bahwa kromosom v_{20} adalah kromosom yang terkuat dan kromosom v_{15} adalah kromosom terlemah.

Seleksi

Prosedur seleksi menggunakan pendekatan cakram rolet. Langkah-langkah untuk membuat cakram rolet adalah sebagai berikut.

1. Hitung nilai *fitness* $Eval(v_k)$ untuk masing-masing kromosom

   $v_k : Eval(v_k) = f(x), \quad k = 1,2,..., pop\_size$

2. Hitung *fitness* total untuk populasi:

   $$F = \sum_{k=1}^{pop\_size} Eval(v_k)$$

3. Hitung kemungkinan seleksi p_k untuk masing-masing kromosom v_k :

$$p_k = \frac{Eval(v_k)}{F}, \quad k = 1,2,..., pop\_size$$

4. Hitung kemungkinan keseluruhan q_k untuk masing-masing kromosom v_k :

   $$q_k = \sum_{j=1}^{k} p_j, \quad k = 1,2,...pop\_size$$

Proses penyeleksian dimulai dengan memutar cakram rolet sebanyak ukuran populasinya. Satu kromosom diseleksi untuk menghasilkan sebuah populasi baru (populasi induk). Langkah-langkahnya adalah sebagai berikut.

1. Bilangan r dibangkitkan secara acak dari *range* [0.1].
2. Jika $r \leq q_1$, kromosom pertama v_1 diseleksi; sebaliknya, seleksi kromosom ke-k (v_k) jika $q_{k-1} < r \leq q_k$, di mana ($2 \leq k \leq$ *pop_size*)

Total *fitness* F dari populasi di atas adalah:

$$F = \sum_{k=1}^{10} Eval(v_k) = 472,725510$$

Peluang seleksi p_k untuk masing-masing kromosom v_k (k = 1,..,20) adalah sebagai berikut:

p_1 =	0.060240	p_8 =	0.049921	p_{15} =	0.029497
p_2 =	0.049204	p_9 =	0.042073	p_{16} =	0.056732
p_3 =	0.049795	p_{10} =	0.056322	p_{17} =	0.051267
p_4 =	0.063301	p_{11} =	0.047634	p_{18} =	0.044206
p_5 =	0.046747	p_{12} =	0.047133	p_{19} =	0.054002
p_6 =	0.035521	p_{13} =	0.042407	p_{20} =	0.068420

p_7 = 0.043513 p_{14} = 0.062065

Peluang kumulatif q_k untuk masing-masing kromosom v_k (k = 1,...,20) adalah sebagai berikut:

q_1	0.060240	q_8	0.398241	q_{15}	0.725373
q_2	0.109444	q_9	0.440314	q_{16}	0.782105
q_3	0.159238	q_{10}	0.496636	q_{17}	0.833372
q_4	0.222539	q_{11}	0.544271	q_{18}	0.877578
q_5	0.269286	q_{12}	0.591404	q_{19}	0.931580
q_6	0.304807	q_{13}	0.633811	q_{20}	1.000000
q_7	0.348320	q_{14}	0.695877		

Sekarang cakram rolet siap diputar sebanyak dua puluh kali dan setiap putaran dilakukan seleksi yang menghasilkan sebuah kromosom induk untuk membentuk sebuah populasi baru. Asumsikan bahwa rangkaian dua puluh angka acak dari *range* [0,1] adalah sebagai berikut:

r_1	0.476736	r_8	0.303807	r_{15}	0.671877
r_2	0.725373	r_9	0.920580	r_{16}	0.445636
r_3	0.695777	r_{10}	0.221239	r_{17}	0.513271
r_4	0.752105	r_{11}	0.420314	r_{18}	0.182539
r_5	0.725173	r_{12}	0.367241	r_{19}	0.107544
r_6	0.050240	r_{13}	0.891580	r_{20}	0.142238
r_7	0.304407	r_{14}	0.312320		

Angka pertama r_1 = 0.476736 lebih besar daripada q_9 dan lebih kecil daripada q_{10}. Hal ini berarti bahwa kromosom v_{10} diseleksi untuk

menghasilkan sebuah populasi baru. Angka kedua r_2 = 0.725373 lebih besar daripada q_{14} dan lebih kecil daripada q_{15}. Hal ini berarti bahwa kromosom v_{15} diseleksi lagi untuk menghasilkan sebuah populasi baru (populasi induk) dan seterusnya. Akhirnya, setelah seleksi selesai dihasilkan sebuah populasi baru (calon kromosom induk), yang terdiri atas kromosom-kromosom berikut ini:

v'_1	[00100110000000100110101001001100 0]	v_{10}
v'_2	[10001100011010100111001111001011 0]	v_{15}
v'_3	[10101010100010111111011101011001 00]	v_{14}
v'_4	[00010111110000111111000100110011 00]	v_{16}
v'_5	[10001100011010100111001111001011 0]	v_{15}
v'_6	[00000110111100000000100111100101 00]	v_1
v'_7	[01000100011110001111001110010010 1]	v_6
v'_8	[01000100011110001111001110010010 1]	v_6
v'_9	[01001000001111101111100011111010 0]	v_{19}
v'_{10}	[10110100010001100010010110111111 01]	v_4
v'_{11}	[11110110010000000001011000101101 11]	v_9
v'_{12}	[00101101011010010011011001010011 0]	v_8
v'_{13}	[01001000001111101111100011111010 0]	v_{19}
v'_{14}	[11000111010001100110110100011111 0]	v_7
v'_{15}	[10101010100010111111011101011001 00]	v_{14}
v'_{16}	[00100110000000100110101001001100 0]	v_{10}
v'_{17}	[10110010100001110110111110100001 10]	v_{11}
v'_{18}	[10110100010001100010010110111111 01]	v_4
v'_{19}	[10110101010011010011111110100101 0]	v_2
v'_{20}	[00100110011111101100011110101000 00]	v_3

Penyilangan (*Crossover*)

Metode *crossover* yang digunakan adalah penyilangan satu titik (*one-point crossover*). Dengan metode ini, algoritma genetika akan menyeleksi secara acak satu titik potong dan saling menukarkan bagian

dari dua induk untuk menghasilkan keturunan. Prosedur penyilangannya adalah sebagai berikut:

```
Prosedur : Penyilangan
begin
  k ←0;
  while (k ≤ 10) do
  r_k ←nilai acak dari [0,1];
  if (r_k < 0.25) then
    seleksi v_k sebagai salah satu induk untuk crossover;
  end
  k ←k+1;
 end
end
```

Peluang penyilangan p_c ditetapkan 0,25. Dengan nilai sebesar 0,25, berarti kita mengharapkan rata-rata 25% kromosom dalam populasi akan mengalami penyilangan. Untuk itu, kita bangkitkan bilangan acak r antara [0,1] pada setiap kromosom dalam populasi yang baru terbentuk; jika r < 0,25, kita silangkan kromosom yang bersangkutan.

Diasumsikan rangkaian bilangan acak yang muncul adalah adalah sebagai berikut:

r_1 = 0.622519	r_8 = 0.912566	r_{15} = 0.772587
r_2 = 0.722539	r_9 = 0.222539	r_{16} = 0.882524
r_3 = 0.825539	r_{10} = 0.242539	r_{17} = 0.222539
r_4 = 0.222533	r_{11} = 0.572252	r_{18} = 0.822512
r_5 = 0.621138	r_{12} = 0.432519	r_{19} = 0.652509
r_6 = 0.612540	r_{13} = 0.482599	r_{20} = 0.222539

r_7 = 0.922512 r_{14} = 0.322536

Hal tersebut mengandung arti bahwa kromosom-kromosom v'_9 dan v'_{10} ' diseleksi untuk *crossover*. Oleh karena 33 adalah panjang total dari sebuah kromosom, sebuah *pos* bilangan integer akan dibangkitkan secara acak dalam *range* [1,32]. Titik *pos* ini digunakan sebagai titik potong (titik *crossover*). Diasumsikan bahwa *pos* bilangan yang dihasilkan adalah 4. Dua kromosom dipotong setelah bit ke-empat, yaitu:

v'_9 = [0100 | 10000011111011111000111110100]

v'_{10} = [1011 | 01000100011000100101101111101]

Keturunan yang dihasilkan dengan cara saling menukarkan bagian kiri dari induk mereka menjadi sebagai berikut:

v''_9 = [1011 **10000011111011111000111110100**]

v''_{10} = [**0100** 01000100011000100101101111101]

Mutasi

Proses mutasi akan mengganti satu atau lebih gen dengan peluang yang besarnya sama dengan nilai mutasi. Asumsikan bahwa gen ke-18 dari kromosom v_9 diseleksi untuk sebuah mutasi. Oleh karena gen ke-3 adalah 1, gen tersebut akan diubah menjadi 0. Jadi kromosom setelah mengalami mutasi akan menjadi:

v'_9 = [10111000001111101111100011111010 0]

v''_9 = [10011000001111101111100011111010 0]

Peluang mutasi (p_m) ditetapkan bernilai 0,01. Hal ini mengandung arti bahwa kita mengharapkan rata-rata 1% dari total bit dalam populasi akan mengalami mutasi. Dalam contoh ini ada m x ukuran populasi yaitu

33 x 20 = 660 bit. Dengan demikian diharapkan akan terjadi 6,6 kali mutasi tiap generasi.

Setiap bit memiliki kesempatan yang sama untuk mengalami mutasi. Jadi, perlu dibangkitkan serangkaian bilangan acak r_k (k = 1,...,660) dalam rentang [0,1]. Anggap bahwa gen-gen berikut ini mengalami mutasi.

Bit_pos	Chrom_Num	Bit_No	Random_Num
105	4	6	0,009797
262	8	32	0,003224
299	10	1	0,000923
420	13	32	0,002272
455	14	7	0,000825
501	16	24	0,001928
583	18	18	0,003924

Populasi akhir (generasi ke-1) yang didapatkan setelah mutasi adalah sebagai berikut:

v''_1 = [00100110000000100110101001001000 0]
v''_2 = [10001100011010100111001111001011 0]
v''_3 = [10101010100010111111000100110011 00]
v''_4 = [00010111110000111111011101011001 00]
v''_5 = [10001110111100000001001111001010 0]
v''_6 = [00000100011010100111001111001011 0]
v''_7 = [01000100011110001111001110010010 1]
v''_8 = [01000100011110001111001110010010 1]
v''_9 = [01001000001111101111100011111010 1]
v''_{10} = [10110100010001100010010110111100]
v''_{11} = [11110110010000000010110001011011 1]
v''_{12} = [00101101011010010011011001010011 0]
v''_{13} = [01001000001111101110110100011111 0]

v''_{14} = [11000111010001100111100011111010 0]
v''_{15} = [10101010100010100110101001001100 0]
v''_{16} = [00100110000000111110111010110010 0]
v''_{17} = [10110010100000100010010110111111 01]
v''_{18} = [10110100010001110110111101000011 0]
v''_{19} = [10100110011111011000111101010000 0]
v''_{20} = [00110101010011010011111110100101 0]

Nilai yang cocok dari variabel-variabel [x_1, x_2] dan nilai *fitness*nya adalah sebagai berikut:

Eval (v''_1) = f (-0.758067 ; 5.223130) = 26.610404 $
Eval (v''_2) = f (5.282325 ; 5.475793) = 13.943805
Eval (v''_3) = f (7.059582 ; 5.013736) = 30.115179
Eval (v''_4) = f (-1.598252 ; 5.340382) = 26.043191
Eval (v''_5) = f (5.431111 ; 4.625663) = 21.985657
Eval (v''_6) = f (-2.739581 ; 5.475793) = 16.388676
Eval (v''_7) = f (1.038775 ; 5.469930) = 16.791618
Eval (v''_8) = f (1.038775 ; 5.469930) = 16.791618
Eval (v''_9) = f (1.261349 ; 5.613538) = 25.540347
Eval (v''_{10}) = f (7.633357 ; 5.102557) = 29.907709
Eval (v''_{11}) = f (11.524958 ; 5.278280) = 19.889001
Eval (v''_{12}) = f (-0.321500 ; 5.543031) = 23.598764
Eval (v''_{13}) = f (1.261349 ; 5.298566) = 20.843833
Eval (v''_{14}) = f (8.754122 ; 5.613486) = 25.254296
Eval (v''_{15}) = f (7.059236 ; 5.223545) = 31.484485
Eval (v''_{16}) = f (-0.757721 ; 5.340382) = 24.460976
Eval (v''_{17}) = f (7.529212 ; 5.102609) = 25.035051 $
Eval (v''_{18}) = f (7.633645 ; 5.355428) = 27.297464
Eval (v''_{19}) = f (6.820360 ; 4.506751) = 18.079987
Eval (v''_{20}) = f (0.143925 ; 5.790609) = 18.418083

Tahapan di atas adalah tahapan **Generasi pertama** dalam proses algoritma genetika, yang berakhir setelah generasi ke-1.000. Dalam eksperimen yang telah dilakukan oleh penulis, jumlah populasi yang digunakan adalah 20, panjang kromosom 33, peluang penyilangan 0,90

dan peluang mutasi 0,30. Hasil eksperimen tersebut dituangkan dalam bentuk grafik seperti pada Gambar 3.5 yaitu mencakup nilai *fitness* maksimum dan nilai *fitness* rata-rata dari tiap generasi. Selama proses iterasi itu, ditemukan kromosom terbaik yaitu pada generasi ke-500, dengan nilai *fitness* **38,849309**. Struktur kromosom tersebut adalah sebagai berikut:

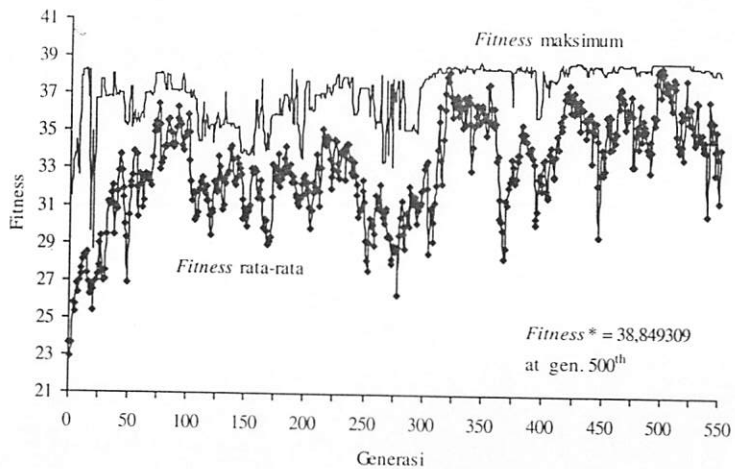
v^* = [11110111111110000111110100101011 1]

Nilai $x1$ dan $x2$ yang optimal yaitu:

$x_1^* = 11.626395$

$x_2^* = 5.724876$

$f(x_1^*, x_2^*) = \mathbf{38.849309}$



Gambar 3.5 Grafik hasil eksperimen terhadap fungsi sulit pada kasus 2

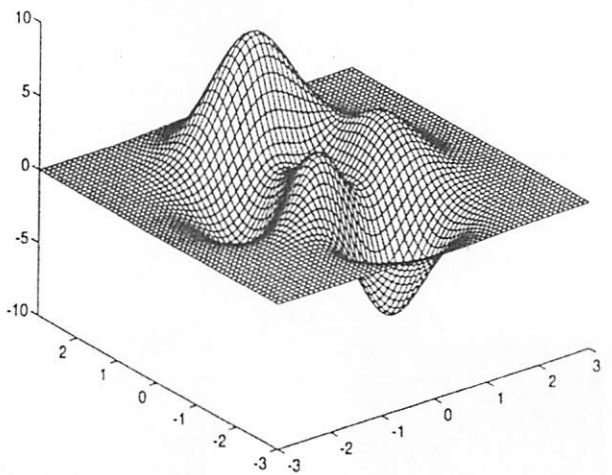
3.3 Kasus 3: Optimasi Fungsi Sulit-2

Seperti pada Kasus 2, contoh pada Kasus 3 ini juga akan diperlihatkan bagaimana algoritma genetika dapat dipakai untuk menyelesaikan optimasi fungsi sulit-berkendala. Fungsi ini telah dipakai oleh Chong dan Zak (1996) yaitu sebagai berikut:

$$f(x,y) = 3(1-x)^2 e^{-x^2-(y+1)^2} - 10\left(\frac{x}{2} - x^3 - y^4\right)e^{-x^2-y^2} - \frac{e^{-(x+1)^2-y^2}}{3}$$

$$-3 \le x, y \le 3$$

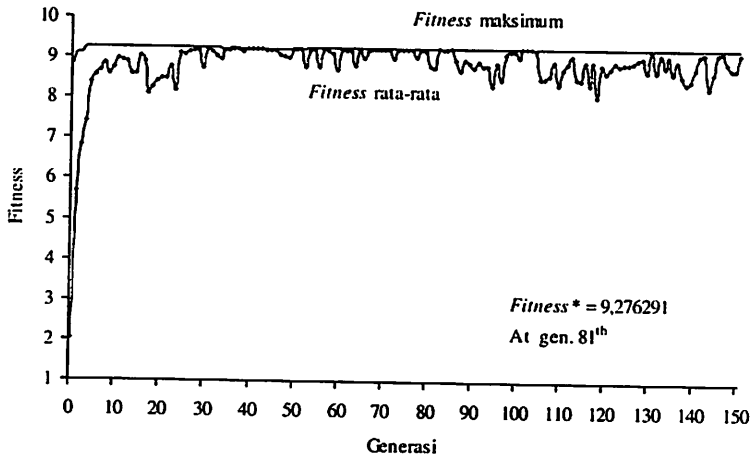
Berdasarkan eskperimen yang telah dilakukan oleh Chong dan Zak, keduanya menemukan bahwa solusi optimumnya adalah 8,0926 dengan nilai x= -0,004944 dan y=1,645203. Adapun visualisasi fungsi tersebut dapat dilihat pada Gambar 3.6.



Gambar 3.6 Grafik fungsi sulit-2 untuk Kasus 3 (Chong dan Zak 1997)

Sementara itu, dalam ekperimen yang telah kami lakukan dalam bahasa Pascal, didapat solusi optimalnya adalah 9,276291 (yaitu pada

generasi ke-81) dengan nilai variabelnya yaitu x*=-0,0303 dan y*=1,5455. Dari hasil ini, disimpulkan bahwa algorima genetika mampu menemukan solusi optimum yang tidak dapat diselesaikan dengan metode-metode konvensional. Tampilan grafik hasil eksperimen yang telah kami lakukan dapat dilihat pada Gambar 3.7. Sementara itu, hasil *running* programnya, dapat dilihat pada halaman selanjutnya.



Gambar 3.7 Grafik nilai *fitness* maksimum dan *fitness* rata-rata untuk kasus 4

INITIAL REPORT

Pop.Size = 20 ; Chrom.Length = 32; Max.Generation = 1000; Crossover probability = 0.8800; Mutation probability = 0.1000

	< Chromosome >	Decimal 1 Length=16)	Decimal 2 (Length=16)	x1	x2	Fitness
1)	11010011000010100110110010110110	54026	27830	1.946304	-0.452049	1.233727
2)	01000001111111111011111000101010011	16895	48211	-1.453193	1.413916	0.240874
3)	10010110110000000010000011100100	38592	8420	0.533257	-2.229114	1.393679
4)	10001100110110101001001100110100	36461	19252	0.338155	-1.237400	5.367060
5)	10110001000101110110000011000011	45335	24963	1.150607	-0.714534	1.944950
6)	10010001011011000011101111011001	37228	15321	0.408377	-1.597299	4.824775
7)	01110101100000001010111010110101000	30081	23912	-0.245960	-0.810758	6.766873
8)	01110010001001101100111111110101010	29222	53226	-0.324605	1.873060	3.346748
9)	11010101101011111111011011011001	54703	60889	2.008286	2.574640	0.011952
10)	00010010011100011101011001101001	4721	54889	-2.567773	2.025315	-0.000199
11)	00101110001010111010100000101010	11819	43050	-1.917922	0.941405	-0.597280
12)	01011100001011111100010111010001	24087	58089	-0.794736	2.318288	0.707682
13)	01000101011011100000010000011111	17774	1055	-1.372717	-2.903410	0.091390
14)	01111001010011000100101000010010	31052	18962	-0.157061	-1.263951	8.808600
15)	11000010011011110001001010110110	49775	6486	1.557107	-2.406180	0.110282
16)	10010011001010110101111111011000	37675	24536	0.449302	-0.753628	1.550087
17)	10101101000000110101101100000101	44291	23301	1.055024	-0.866697	1.877935
18)	01101101111000011110010011100000	28401	58608	-0.399771	2.365805	0.993754
19)	11110110101001011111011001110001	63141	63281	2.780819	2.793637	0.000145
20)	10110111101010001100100100011110	47028	25678	1.305608	-0.649073	2.132440

Sum of Fitness = 40.805476
Max. Fitness = 8.808600
Min. Fitness = -0.597280
Avg. Fitness = 2.040274

Generation 80 and generation 81

	< Chrom. >	<Decimal_1>	<Decimal_2)	x	y	< Fitness >	
1)	01110010110011000100101000001010	29388	18954	-0.309407	-1.264683	9.241657	
2)	01110010100011000100101000001010	29324	18954	-0.315267	-1.264683	9.248569	
3)	01111010100011000100101000001010	31372	18954	-0.127764	-1.264683	8.675279	$
4)	01100010110011000100101000001010	25292	18954	-0.684413	-1.264683	8.169127	
5)	01110010110001000100101000001010	29380	18954	-0.310140	-1.264683	9.242562	$
6)	01110010000011000100101000001010	29196	18954	-0.326986	-1.264683	9.260110	
7)	01110010110011000100101000001010	29388	18954	-0.309407	-1.264683	9.241657	
8)	01110000100011000100101000001010	28812	18954	-0.362142	-1.264683	9.276284	
9)	01110010100011000100101000001010	29324	18954	-0.315267	-1.264683	9.248569	
10)	01110011000011000100101000001010	29452	18954	-0.303548	-1.264683	9.233988	
11)	01110010100011000100101000001010	29324	18954	-0.315267	-1.264683	9.248569	
12)	01110010100011000100101100001010	29324	19210	-0.315267	-1.241245	9.237559	$
13)	01110111100011000100101000001010	30604	18954	-0.198077	-1.264683	8.971200	
14)	01110000100011000100101000001010	28812	18954	-0.362142	-1.264683	9.276284	
15)	01110010100011000100101000001010	29324	18954	-0.315267	-1.264683	9.248569	
16)	01110010100011000100101000001010	29324	18954	-0.315267	-1.264683	9.248569	
17)	01110010100011000100101000001010	29324	18954	-0.315267	-1.264683	9.248569	
18)	01110000000011000100101000001010	28684	18954	-0.373861	-1.264683	9.275476	
19)	01110010000011000100101000001010	29196	18954	-0.326986	-1.264683	9.260110	
20)	01100010110011001100101000001010	25292	51722	-0.684413	1.735363	2.789009	$

	<P1 , P2>	<Site>	< Ch1&Ch2 >	<Decimal_1>	<Decimal_2>	x	y	< Fitness >	
1)	(1, 2)	9	01110010110011000100101000001010	29388	18954	-0.309407	-1.264683	9.241657	
2)	(1, 2)	9	01110010100011000100101000001010	29324	18954	-0.315267	-1.264683	9.248569	
3)	(14,17)	20	01110000100011000100101000001010	28812	18954	-0.362142	-1.264683	9.276284	
4)	(14,17)	20	01110010100011000100101000001010	29324	18954	-0.315267	-1.264683	9.248569	
5)	(14, 7)	0	01110000100011000100101000001010	28812	18954	-0.362142	-1.264683	9.276284	
6)	(14, 7)	0	01110010110011000100101000001010	29388	18954	-0.309407	-1.264683	9.241657	
7)	(9, 7)	30	01110010110011000100101000001010	29388	18954	-0.309407	-1.264683	9.241657	
8)	(9, 7)	30	01110010100011000100101000001010	29324	18954	-0.315267	-1.264683	9.248569	
9)	**(5, 8)**	**31**	**01110000100011000100101000011010**	**28812**	**18970**	**-0.362142**	**-1.263218**	**9.276291**	**$**
10)	(5, 8)	31	01110010110000000100101000001010	29376	18954	-0.310506	-1.264683	9.243011	$
11)	(14, 3)	9	01110000100011000100101000001010	28812	18954	-0.362142	-1.264683	9.276284	
12)	(14, 3)	9	01111010100011000100101000001010	31372	18954	-0.127764	-1.264683	8.675279	
13)	(1,10)	4	01110010110011000100101000001010	29388	18954	-0.309407	-1.264683	9.241657	
14)	(1,10)	4	01110011000011000100101000001010	29452	18954	-0.303548	-1.264683	9.233988	
15)	(18,18)	15	01110000000011000100101000001010	28684	18954	-0.373861	-1.264683	9.275476	
16)	(18,18)	15	01110000000011000100101000001010	28684	18954	-0.373861	-1.264683	9.275476	
17)	(10, 2)	0	01110011000011000100101000001010	29452	18954	-0.303548	-1.264683	9.233988	
18)	(10, 2)	0	01110010100011000100101000001010	29324	18954	-0.315267	-1.264683	9.248569	
19)	(18, 7)	0	00110000000011000100101000001010	12300	18954	-1.873884	-1.264683	0.472235	$
20)	(18, 7)	0	01110010010011000100101000001010	29260	18954	-0.321126	-1.264683	9.254721	$

```
SumFitness Value of generation      81 = 175.730217
Max. Fitness Value of generation    81 =   9.276291
Min. Fitness Value of generation    81 =   0.472235
Avg. Fitness Value of generation    81 =   8.786511
Cum. of NCross until gen. 81 = 709
Cum. ofNMutation until gen. 81 = 152
```


Bab 4
Bagaimana Algoritma Genetika Bekerja?

Hingga saat ini, algoritma genetika telah terbukti "ampuh" untuk memecahkan masalah optimasi yang rumit dan berskala besar. Algoritma genetika dapat menemukan solusi masalah itu secara efisien dengan hanya menggunakan operasi-operasi genetik yang sederhana saja (penyilangan, mutasi, dan seleksi). Namun, bagaimana sebenarnya algoritma genetika dapat melakukan semua itu? Bagaimana cara kerjanya? Bagaimana penjelasan teoritisnya? Bisakah teori itu dijelaskan secara matematis?

Pertanyaan-pertanyaan di atas bukanlah pertanyaan aneh. Justru pertanyaan itulah yang akan menuntun kita untuk memahami bagaimana sebenarnya algoritma genetika bekerja dan dapat melakukan semua itu. Bab 4 buku ini akan mencoba menjawab pertanyaan-pertanyaan di atas.

4.1 Teori Schemata

Teori yang pertama kali menjelaskan bagaimana algoritma genetika bekerja adalah Teori *Schemata*. Teori ini dicetuskan oleh John Holland tahun 1975. Saat itu, Holland menerbitkan buku *Adaptation in Natural and Artificial Systems* yang di dalamnya terdapat analisis matematis tentang cara kerja algoritma genetika. Untuk memberikan penjelasan tentang Teori Schemata, penulis mengacu pada penjelasan

yang telah ditulis oleh Goldberg (1989). Berikut adalah penjelasannya secara detail.

Istilah *schema* (jamaknya: *schemata*) berasal dari bahasa Yunani yang artinya "bentuk" atau "rancangan". *Schema* dapat diartikan sebagai rancangan umum beberapa kromosom yang memiliki kesamaan. Dengan kata lain, sebuah *schema* adalah sebuah representasi dari beberapa unit kromosom yang memiliki kemiripan. Kemiripan ini dilihat dari *fitness*-nya maupun strukturnya. Untuk menjelaskan Teori *Schemata* seperti yang dicetuskan oleh Holland, dalam buku ini akan dipakai beberapa istilah dan persamaan matematis yang banyak melibatkan unsur probabilitas. Pertama, istilah *schema* dipakai untuk merujuk pada beberapa unit kromosom yang memiliki kemiripan. Kedua, dipakai simbol asterik "*" yang disebut simbol "*don't care*". Simbol ini digunakan jika nilai bit dalam kromosom tidak begitu dipedulikan (nilainya bisa 0 atau 1). Sebagai contoh, terdapat dua buah kromosom yaitu [1001] dan [1101]. Kedua kromosom ini mempunyai kemiripan nilai bit pada posisi ke-1, ke-3, dan ke-4. Dengan demikian, *schema* yang dapat mewakili kemiripan kedua kromosom ini adalah [1*01]. Contoh *schema* yang lainnya yaitu 0*101*. *Schema* ini dapat mewakili empat unit kromosom yaitu kromosom [001010], [001011], [011010], dan [011011]. Kemiripan kromosom-kromosom tersebut terletak pada nilai bit ke-1, ke-3, ke-4, dan ke-5.

Dari kedua contoh *schema* di atas, dapat dikatakan bahwa proses pencarian solusi optimum yang dilakukan oleh algoritma genetika, tidak lain dan tidak bukan, adalah proses "meningkatkan" jumlah kromosom dalam populasi agar semuanya mirip dengan *schema* tertentu yang nilai *fitness*-nya sangat baik. Proses perbaikan *schema* inilah yang menjadi kunci bagaimana sebenarnya algoritma genetika bekerja.

4.2 Analisis Matematis Teori *Schemata*

Dalam analisis matematis Teori *Schemata* adalah sebagai berikut. Misalkan kita memiliki sebuah *schema*. *Schema* ini kita harapkan nilai *fitness*-nya meningkat seiring dengan pertambahan jumlah generasi *k*. Peningkatan nilai *fitness* ini dipengaruhi oleh beberapa faktor yaitu antara lain oleh proses seleksi, penyilangan, dan mutasi.

4.2.1 Pengaruh Proses Seleksi terhadap *Schemata*

Ide awal yang dapat dijadikan sebagai pedoman bahwa algoritma genetika benar-benar bekerja (menemukan solusi optimal) adalah: jika sebuah *schema* memiliki nilai *fitness* di atas rata-rata, kromosom-kromosom yang direpresentasikannya akan memiliki peluang terseleksi untuk *survive* lebih besar dibandingkan dengan kromosom-kromosom yang tidak direpresentasikannya.

Untuk menguantifikasi pedoman di atas, kita memerlukan beberapa notasi tambahan. Misalkan H adalah sebuah *schema*, $P(k)$ adalah populasi ke-k, dan $e(H,k)$ adalah jumlah kromosom dalam populasi $P(k)$ yang "*match*" (sesuai/cocok) dengan H. Kemudian, anggap $f(H,k)$ adalah nilai *fitness* rata-rata kromosom dalam populasi $P(k)$ yang bersesuaian dengan schema H. Hal ini berarti bahwa jika $H \cap P(k) = \{x_1,...,x_{e(H,k)}\}$, maka

$$f(H,k) = \frac{f(x_1)+...f(x_{e(H,k)})}{e(H,k)}$$

Perlu diperhatikan bahwa $f(H,k)$ di atas tidak lain dan tidak bukan adalah nilai *fitness* rata-rata dari *schema* H. Sekarang, misalkan N adalah jumlah kromosom dalam populasi k, kemudian $F(k)$ adalah total nilai *fitness* populasi k, dan $\overline{F}(k)$ adalah nilai *fitness* rata-rata kromosom dalam populasi k. Dengan demikian, dapat diturunkan rumus:

$$\overline{F}(k) = \frac{F(k)}{N} = \frac{1}{N}\sum f(x_i^{(k)})$$

Akhirnya, sekarang kita asumsikan $m(H,k)$ adalah jumlah kromosom dalam *mating pool* $M(k)$ yang bersesuaian dengan *schema H*, atau dengan kata lain *m(H,k)* adalah jumlah bit dalam $M(k) \cap H$.

Lemma 4.1

Misalkan H adalah sebuah *schema*, $\mu(H,k)$ adalah peluang $m(H,k)$, maka:

$$\mu(H,k) = \frac{f(H,k)}{\overline{F}(k)} e(H,k)$$

Pembuktian 4.1

Misalkan $P(k) \cap H = \{x_1,...,x_{e(H,k)}\}$. Untuk setiap bit $m^{(k)} \in M(k)$ dan i=1,...,*e(H,k)*, peluang $m^{(k)} = x_i$ didapatkan dengan rumus $f(x_i)/F(k)$. Dengan demikian, peluang jumlah kromosom dalam $M(k)$ yang akan sama dengan x_i adalah:

$$N\frac{f(x_i)}{F(k)} = \frac{f(x_i)}{\overline{F}(k)}$$

Dari uraian di atas, dapat kita peroleh peluang jumlah kromosom dalam $P(k) \cap H$ yang akan terseleksi ke dalam $M(k)$ yaitu sebagai berikut:

$$\sum_{i=1}^{e(H,k)} \frac{f(x_i)}{\overline{F}(k)} = e(H,k)\frac{\sum_{i=1}^{e(H,k)} f(x_i)}{e(H,k)}\frac{1}{\overline{F}(k)}$$

$$= \frac{f(H,k)}{\overline{F}(k)} e(H,k)$$

Oleh karena kromosom dalam $M(k)$ adalah juga kromosom dalam $P(k)$, kromosom dalam $M(k) \cap H$ adalah kromosom-kromosom dalam $P(k) \cap H$ yang terpilih ke dalam $M(k)$. Dengan demikian, terbukti bahwa $m(H,k)$ adalah:

$$\mu(H,k) = \frac{f(H,k)}{\overline{F}(k)} e(H,k)$$

Pembuktian Lemma 4.1 di atas memperkuat pedoman kita bahwa jika sebuah *schema H* memiliki nilai *fitness* lebih baik dari nilai *fitness* rata-rata populasi, peluang jumlah kromosom yang "*match*" dengan *schema H* di *mating pool* $M(k)$ akan lebih besar daripada jumlah kromosom yang ada di populasi $P(k)$.

Perlu diperhatikan bahwa dalam suatu populasi bisa terdapat beberapa *schema*. Masing-masing *schema* memiliki "anggota" tersendiri, yaitu kromosom-kromosom yang berada dalam populasi. Seiring dengan berlangsungnya proses evolusi, anggota masing-masing *schema* ini kemudian akan mengalami penyilangan atau mutasi untuk membentuk suatu *schema* baru yang lebih baik dari sebelumnya. Pembentukan suatu *schema* baru akibat penyilangan dan mutasi akan dibahas berikut ini.

4.2.2 Pengaruh Proses Penyilangan terhadap *Schemata*

Sekarang kita akan menganalisis bagaimana pengaruh proses evolusi terhadap kromosom-kromosom hasil seleksi yang sudah ada di dalam *mating pool*. Untuk hal ini, kita akan menggunakan dua parameter tambahan yang berguna untuk mengenali performa sebuah *schema*. Kedua parameter tersebut adalah *order* (ordo), disimbolkan *o*, dan *distance* (jarak), disimbolkan *d*. Ordo adalah jumlah bit yang *fix* (bukan bersimbol "*don't care*") dari sebuah *schema*. Sedangkan *distance* adalah jarak antara bit paling kiri (yang *fix*) dengan bit paling kanan (yang *fix*). Sebagai contoh, perhatikan *schema* di bawah ini.

$o(1**01) = 4$; dan

$o(0**1*01) = 4$

$d(1**01) = 5 - 1 = 4$

$d(0**1*0*) = 6 - 1 = 5$

Perhatikan bahwa jika sebuah *schema S* dengan panjang *L*, ordo *schema* tersebut $o(S)$ bernilai antara 0 dan *L*, sedangkan *distance*-nya adalah antara 0 dan *L-1*. Jika elemen *schema S* semuanya terdiri atas simbol "*", *distance*-nya akan bernilai nol. Jika elemen *schema S* terdiri atas bit yang *fix*, ordonya bernilai sama dengan *L*. Sebagai contoh, $o(1100101)=7$. Sekarang kita amati bagaimana pengaruh proses penyilangan terhadap sebuah *schema* yang ada di *mating pool*. Ide dasar yang dapat dijadikan pedoman kita adalah: jika ada sebuah kromosom dalam $M(k) \cap H$, peluang kromosom tersebut ada dalam *schema H* akan dibatasi oleh p_c dan $d(H)$

Lemma 4.2

Misalkan ada sebuah kromosom dalam $M(k) \cap H$. Peluang kromosom tersebut disilangkan adalah dan keturunannya ada dalam *schema H*, memiliki hubungan sebagai berikut:

$$p_c = \frac{d(H)}{L-1}$$

Pembuktian

Misalkan ada sebuah kromosom dalam $M(k) \cap H$. Peluang kromosom tersebut mengalami penyilangan adalah p_c. Jika keturunannya tidak ada satu pun yang mewarisi sifat schema H, titik penyilangannya haruslah berada antara simbol fix paling kiri dengan simbol fix paling kanan. Peluang terjadinya hal ini adalah $d(H)/(L-1)$. Dengan demikian, peluang kromosom tersebut mengalami penyilangan dan tidak ada satu pun anaknya yang "*match*" dengan *schema H*, akan didapatkan dengan rumus:

$$p_c \frac{d(H)}{L-1}$$

Dari Lemma 4.2 di atas, dapat kita simpulkan bahwa jika ada sebuah kromosom dalam $M(k) \cap H$, peluang tidak mengalami penyilangan, atau sedikitnya salah satu di antara keturunannya mewakili sifat *schema H*, akan didapatkan dengan rumus:

$$1 - p_c \frac{d(H)}{L-1}$$

Perhatikan bahwa jika ada sepasang kromosom dalam *schema H* yang terpilih untuk disilangkan, secara alami kedua anaknya akan berada dalam *schema H*. Oleh sebab itu, setiap kromosom dalam $M(k) \cap H$ akan memiliki sejumlah peluang tertentu untuk tetap berada dalam *schema H* walaupun telah mengalami penyilangan. Tahap berikutnya dalah mengamati pengaruh mutasi terhadap *schema* yang ada dalam *mating pool*.

4.2.3 Pengaruh Proses Mutasi terhadap *Schemata*

Lemma 4.3.

Misalkan ada sebuah kromosom dalam $M(k) \cap H$. Peluang kromosom ini tetap berada dalam H setelah mutasi dirumuskan sebagai berikut:

$$(1 - p_m)^{o(H)}$$

Pembuktian.

Jika sebuah kromosom dalam $M(k)$, kromosom ini akan tetap dalam H jika dan hanya jika tidak satu pun bit dalam kromosom ini mengalami perubahan akibat mutasi. Peluang kejadian ini adalah $(1 - p_m)^{o(H)}$.

Perhatikan bahwa jika p_m sangat kecil, persaman $(1-p_m)^{o(H)}$ di atas akan sama dengan:

$$1-p_m o(H)$$

Teorema berikut ini akan menggabungkan *lemma-lemma* yang telah diuraikan di atas.

Teorema 4.1

Misalkan sebuah *schema* diberi nama *H*, dan $\varepsilon(H,k+1)$ adalah peluang $\varepsilon(H,k+1)$, yaitu peluang terpilihnya kromosom dalam *schema H* dari suatu populasi *P(k)*. Dengan demikian, dapat dibuat perumusan:

$$\varepsilon(H,k+1) \geq \left(1-p_c \frac{d(H)}{L-1}\right)(1-p_m)^{o(H)}\left(\frac{f(H,k)}{\overline{F}(k)}\right)e(H,k)$$

Pembuktian.

Misalkan $M(k) \cap H$ adalah sebuah *schema H* yang ada dalam *mating pool M* pada generasi *k*. Jika setelah proses evolusi terdapat kromosom yang masih anggota *schema H*, kromosom tersebut adalah $P(k+1) \cap H$. Dengan menggunakan Lemma 4.2 dan Lemma 4.3, peluang terjadinya hal ini:

$$\left(1-p_c \frac{d(H)}{L-1}\right)(1-p_m)^{o(H)}$$

Dengan demikian, karena setiap kromosom dalam $M(k) \cap H$ menghasilkan keturunan dalam $P(k+1) \cap H$ dengan peluang kejadian mengikuti persamaan di atas, peluang $e(H,k+1)$ dalam *mating pool* $M(k)$ adalah:

$$\left(1-p_c \frac{d(H)}{L-1}\right)(1-p_m)^{o(H)} m(H,k)$$

Dari rumusan di atas, didapatkan rumusan baru yaitu:

$$\varepsilon(H,k+1) \geq \left(1-p_c \frac{d(H)}{L-1}\right)(1-p_m)^{o(H)} \mu(H,k)$$

Akhirnya, dengan menggunakan Lemma 4.1, kita akan mendapatkan hasil seperti yang kita inginkan pada Teorema 4.1, yaitu sebagai berikut:

$$\varepsilon(H,k+1) \geq \left(1-p_c \frac{d(H)}{L-1}\right)(1-p_m)^{o(H)} \frac{f(H,k)}{\overline{F}(k)} e(H,k)$$

Teorema 4.1 di atas mengindikasikan bagaimana jumlah kromosom dalam sebuah *schema* berubah-ubah dari generasi ke generasi. Tiga faktor yang memengaruhi perubahan ini adalah: $1-p_c.d(H)/(L-1)$, $(1-p_m)^{o(H)}$ dan $f(H,k)/\overline{F}(k)$. Pengaruh masing-masing faktor adalah sebagai berikut:

- $f(H,k)/\overline{F}(k)$: menunjukkan nilai *fitness* rata-rata dari sebuah *schema H*. Semakin tinggi nilai *fitness* rata-rata *schema H*, semakin tinggi pula peluang jumlah kromosom yang "*match*" pada generasi berikutnya.
- $1-p_c.d(H)/(L-1)$: mengindikasikan pengaruh operator penyilangan. Semakin kecil nilai $p_c.d(H)/(L-1)$, semakin besar peluang jumlah kromosom yang *match* dengan *schema H* pada generasi berikutnya.
- $(1-p_m)^{o(H)}$: mengindikasikan pengaruh mutasi. Semakin besar nilai $(1-p_m)^{o(H)}$, semakin besar pula peluang jumlah kromosom yang "*match*" dengan *schema H* pada generasi berikutnya.

Dari penjelasan ketiga faktor di atas, dapat kita simpulkan bahwa sebuah *schema* dikatakan cukup baik (menjanjikan) apabila *schema* itu berordo kecil (*low order*), ber-*distance* rendah (*short*), dan memiliki nilai *fitness* rata-rata yang lebih baik dari *fitness* rata-rata populasi. Kromosom anggota *schema* tersebut akan bertambah dan terus bertambah seiring dengan peningkatan jumlah generasi.

Perlu diperhatikan bahwa analisi matematis teori *schemata* di atas hanya berlaku pada kromosom dengan representasi biner. Untuk kromosom non-biner, teori *schemata* belum dapat menjelaskan bagaimana sebenarnya cara kerja algoritma genetika yang begitu pintar ini hingga sanggup menemukan solusi optimum, meski dalam ruang pencarian begitu komplek. Untuk itu, nampaknya diperlukan teori-teori baru yang sanggup menjelaskannya.

Bab 5
Aplikasi Algoritma Genetika untuk Penjadwalan *Flow-Shop* Bidang Agroindustri

5.1 Permasalahan

Masalah penyusunan jadwal produksi telah menjadi perhatian utama para praktisi yang berkecimpung dalam dunia industri, baik industri manufaktur maupun industri berbasis pertanian (agroindustri). Salah satu penyebabnya yaitu adanya kesulitan menemukan teknik yang tepat untuk membuat jadwal produksi yang paling baik, paling optimal, dan dapat memenuhi segala kriteria-kriteria penjadwalan yang telah ditetapkan. Teknik-teknik penyusunan jadwal produksi yang sudah ada (teknik konvensional) tidak dapat dipakai lagi karena teknik-teknik tersebut memiliki banyak kelemahan, terutama apabila dihadapkan pada masalah berskala besar dan komplek.

Masalah penyusunan jadwal produksi dapat dikatakan sebagai masalah pencarian (*searching*) suatu jadwal yang dapat mengoptimalkan parameter-parameter penjadwalan produksi (nilai optimum) dalam suatu ruang pencarian (*search space*) tertentu. Suatu jadwal optimum akan sulit dicari apabila ruang pencariannya tergolong komplek (*noisy*), seperti bersifat multimodal, tidak kontinu, dan memiliki banyak parameter pengukuran tingkat keoptimalan. Dihadapkan pada masalah seperti itu, teknik-teknik pencarian (*searching techniques*) konvensional tidak dapat digunakan lagi karena

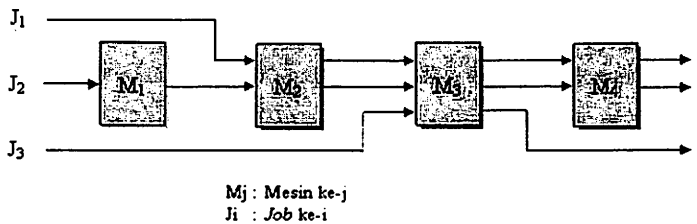
memiliki banyak kelemahan. Dalam kondisi seperti itulah diperlukan teknik yang lebih tangguh, lebih adaptif, dan lebih efisien.

Algoritma genetika termasuk teknik pencarian terbukti bersifat *robust* (tangguh), adaptif, dan efisien. Dalam pencariannya, algoritma genetika bekerja dengan cara meniru proses evolusi dan perubahan struktur genetik pada makhluk hidup. Algoritma genetika sangat cocok untuk memecahkan masalah optimasi yang komplek (*complex optimization problems*), yang sukar atau tidak dapat dipecahkan dengan menggunakan teknik-teknik pencarian dan optimasi konvensional, seperti metode *calculus-based* atau *enumeratif-search*. Kesukaran terjadi karena teknik-teknik konvensional tersebut sangat tidak efisien, bergantung pada adanya fungsi turunan, serta tidak dapat menangani masalah berskala besar.

Salah satu masalah yang tergolong komplek dan bersifat kombinatorial adalah masalah penjadwalan produksi tipe *flow-shop* berskala besar. Dalam tipe ini, ada sejumlah *m job* yang harus dijadwalkan pada sejumlah *n* mesin, di mana setiap *job* memiliki urutan pemrosesan yang sama pada setiap mesin. Tujuan penjadwalannya adalah meminimumkan *makespan* (waktu penyelesaian proses secara keseluruhan). Penjadwalan akan sangat sukar dipecahkan apabila jumlah *job* dan mesin sangat banyak. Teknik-teknik penjadwalan konvensional seperti Aturan Johnson (*Johnson's Rule*) tidak dapat digunakan karena teknik tersebut hanya dapat dipakai untuk dua buah mesin dengan sejumlah m *job*. Teknik enumeratif juga tidak efisien dipakai karena harus menguji semua alternatif penjadwalan yang ada. Untuk dapat mengefisiensikan pencarian jadwal terbaik secara signifikan, digunakanlah algoritma genetika.

5.2 Formulasi Permasalahan

Salah satu ciri utama sistem produksi tipe *flow-shop* adalah adanya lini proses produksi satu arah yang berorientasi pada produk akhir (*product-uriented*). Dalam sistem ini diperlukan beberapa mesin atau fasilitas produksi yang berfungsi khusus (*specific-purpose*) untuk menunjang terciptanya produk akhir. Mesin-mesin atau fasilitas yang berfungsi sama dikelompokkan dalam satu grup. Black (1991) menyebutkan bahwa sistem ini disebut juga sistem produksi massal (*mass production*) jika volume produksinya sangat besar. Skema sistem produksi tipe *flow-shop* dapat dilihat pada Gambar 5.1.



Gambar 5.1 Ilustrasi sistem produksi tipe *flow-shop* (Kusiak 1990)

Gen dan Cheng (1997) mendeskripsikan karakteristik masalah penjadwalan *flow-shop* secara umum yaitu sebagai berikut: 1) ada sejumlah *m job* yang akan diproses pada sejumlah *n* mesin; 2) waktu pemrosesan untuk *job i* pada mesin *j* dinotasikan sebagai t_{ij} ($i=1,...,m$; $j=1,.., n$); 3) setiap mesin hanya dapat memroses satu *job* pada satu waktu tertentu; 4) setiap *job* hanya dapat diproses di satu mesin pada satu waktu tertentu; 5) urutan pemrosesan *job* di setiap mesin adalah sama; 6) tujuan penjadwalannya adalah mencari urutan pemrosesan *job* yang dapat meminimumkan *makespan*.

Berdasarkan ada tidaknya kendala, masalah penjadwalan *flow-shop* dapat dibagi menjadi dua tipe yaitu *unconstrained flow-shop* (*flow-shop* tanpa kendala) dan *constrained flow-shop* (*flow-shop*

dengan kendala). Dalam tipe *unconstrained flow-shop*, tidak ada kendala-kendala yang dapat menghambat proses implementasi *job* pada setiap mesin produksi sehingga semua alternatif jadwal bersifat legal. Sebagai contoh, dalam buku *Feed Manufacturing Technology IV* (McEllhiney 1994), suatu industri pakan ternak akan memproduksi empat macam pakan ternak yaitu pakan ayam, itik, ikan, dan burung. Setiap pakan yang diproses harus melalui mesin pencampur (*mixer*). Masalah yang timbul adalah menentukan urutan pakan yang akan diproses (*job sequence*).

Contoh masalah produksi pakan ternak di atas tergolong masalah *flow-shop*. Jika dalam proses produksi pakan ternak tidak ada kendala, dalam hal reaksi-reaksi kimia sisa pakan ternak sebelumnya yang ada di *mixer* dengan pakan ternak baru yang akan diproses, masalah ini dikategorikan sebagai *unconstrained flow-shop*. Dengan demikian, semua urutan *job* adalah bersifat legal. Artinya, proses produksi dapat dimulai dari pakan ternak mana saja, apakah dimulai dari pakan ikan, pakan itik, pakan ayam, lalu diakhiri pakan burung, atau urutan lainnya Banyaknya jumlah alternatif urutan yang mungkin ini ditentukan oleh jumlah permutasi *job*, yaitu jumlah pakan ternak yang akan diproduksi. Dalam contoh ini, ada 16 (4!=16) alternatif jadwal yang mungkin.

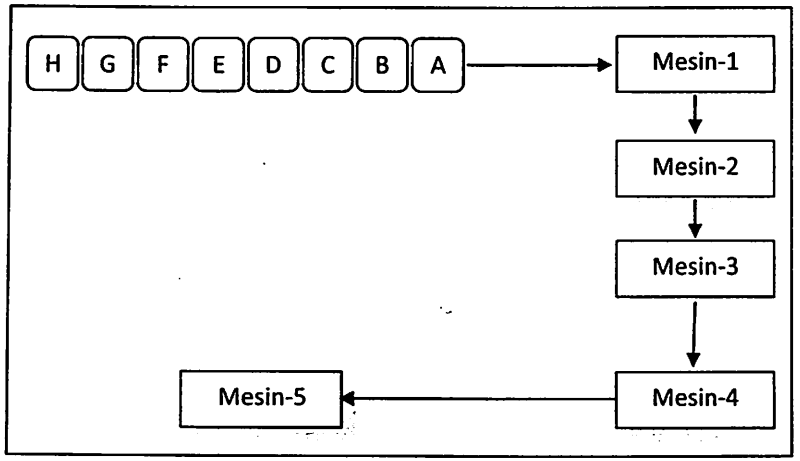
5.3 Representasi Kromosom

Algoritma genetika bekerja pada sekumpulan calon solusi yang dinamakan populasi. Setiap calon solusi (individu) dalam populasi direpresentasikan dalam bentuk kromosom. Representasi kromosom yang dipakai untuk memecahkan masalah penjadwalan *flow-shop* adalah *orde-based representation*. Dalam representasi jenis ini, setiap elemen kromosom menunjukkan suatu *job* tertentu yang spesifik, dan urutan elemen menunjukkan urutan pemrosesan *job* pada mesin-mesin produksi. Sebagai contoh, proses produksi seperti pada Gambar 5.2 yang

terdiri atas 8 macam *job* (A, B, C, D, E, F, G, dan H) yang akan diproses pada 5 buah mesin, representasi suatu kromosom V_t akan berbentuk:

$$V_t = [\,A\ B\ C\ D\ E\ F\ G\ H\,]$$

di mana A, B, C, D, E, F, G, dan H adalah *job-job* yang akan diproses dengan urutan dimulai dari *job* A dan diakhiri oleh *job* H.



Gambar 5.2 Skema sistem produksi tipe *flow-shop*

5.4 Fungsi Tujuan (Fungsi *Fitness*)

Setelah menentukan representasi kromosom, langkah selanjutnya untuk dapat mengaplikasikan algoritma genetika adalah menentukan fungsi evaluasi kromosom (fungsi *fitness*) berdasarkan fungsi tujuan yang telah ditetapkan. Fungsi tujuan dalam masalah penjadwalan *flow-shop* adalah meminimumkan *makespan* (waktu penyelesaian *job* secara keseluruhan), yaitu:

$$Min \sum_{j=1}^{n} \sum_{i=1}^{m} t_{ij} \qquad \text{..... (5.1)}$$

Dimana : m = jumlah *job*
n = jumlah mesin
t_{ij} = waktu pemrosesan *job*-i di mesin-j

Dengan menggunakan persamaan (5.1), nilai *makespan* kromosom ke-p (Z_p) dapat dihitung menggunakan rumus:

$$Z_p = \sum_{j=1}^{n} \sum_{i=1}^{m} t_{ij} \qquad \text{..... (5.2)}$$

Oleh karena masalah penjadwalan *flow-shop* adalah masalah minimasi *makespan*, nilai fungsi tujuan setiap kromosom harus dikonversi ke dalam nilai *fitness* sehingga kromosom yang lebih *fit* (lebih bugar) akan memiliki nilai *fitness* yang lebih tinggi. Konversi dilakukan dengan menggunakan fungsi Persamaan 5.3

$$\Phi_p = \frac{1}{Z_p} \qquad \text{..... (5.3)}$$

Di mana Φ_p adalah nilai *fitness* kromosom ke-p dalam populasi.

5.5 Penyilangan dan Mutasi

Dalam masalah *flow-shop* ini, operator penyilangan yang digunakan adalah *Partially-Mapped Crossover* (PMX). Prosedur penyilangan PMX ini yaitu dimulai dengan menentukan dua buah titik penyilangan secara acak pada sepasang kromosom induk. Setelah itu, elemen kromosom yang terletak di antara kedua titik tersebut saling dipertukarkan untuk membentuk dua buah kromosom anak. Kromosom anak yang baru terbentuk kemudian diperiksa kelegalannya. Apabila

kromosom tersebut legal (memiliki elemen yang sama), kromosom tersebut dilegalkan berdasarkan kaidah pemetaan. Kaidah ini diperoleh dari hasil penukaran elemen antarkromosom induk.

Operator mutasi yang digunakan dalam masalah ini adalah *reciprocal exchange mutation*, yang merupakan bagian dari *swap mutation*. Prosedur mutasinya yaitu dimulai dengan menentukan sebuah posisi secara acak pada kromosom anak yang baru terbentuk hasil proses penyilangan. Elemen kromosom pada posisi ini kemudian dipertukarkan dengan elemen yang ada di sebelah kanannya. Apabila posisi acak yang diperoleh terletak pada posisi akhir kromosom, penukaran elemen dilakukan dengan elemen yang berada pada posisi awal.

5.6 Contoh Penerapan Algoritma Genetika

5.6.1 Kasus 1: 4 *job* – 2 mesin

Masalah penjadwalan 4 *job*-2 mesin adalah masalah penentuan urutan pemrosesan *job* terbaik (dari 4 *job* yang ada) pada 2 buah mesin. Urutan pemrosesan yang mungkin ada 24 alternatif (4! = 24). Dengan demikian, besarnya ruang pencarian adalah 24 unit jadwal—yang direpresentasikan 24 unit kromosom.

Kasus nyata (*real case*) untuk masalah penjadwalan 4 *job*-2 mesin terdapat pada industri pakan ternak (McEllhiney 1994). Dalam industri ini, terdapat empat jenis pakan ternak yang berbeda yang akan diproduksi pada dua buah mesin utama yaitu mesin pencampur (*mixer*) bahan baku dan mesin pelet. Setiap bahan baku pakan ternak hasil penggilingan harus melalui mesin pencampur kemudian dilanjutkan di mesin pelet. Tujuan penjadwalannya adalah menentukan urutan pakan ternak yang akan diproses pada kedua mesin tersebut. Waktu pemrosesan *job* untuk Kasus 1 ini dapat dilihat pada Tabel 5.1.

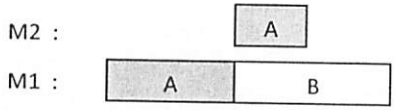
Tabel 5.1 Waktu pemrosesan *job* di setiap mesin untuk kasus 1

Mesin\Job	A	B	C	D
M1	10	14	4	20
M2	6	8	16	12

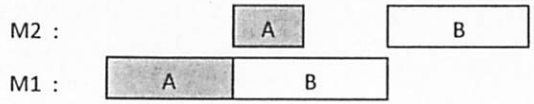
Kasus 1 ini digunakan untuk menguji kebenaran program algoritma genetika yang dikembangkan. Pengujian dilakukan dengan membandingkan nilai *makespan* hasil hitungan secara enumeratif dengan nilai *makespan* hasil *output* program. Contoh penghitungan nilai *makespan* secara enumeratif untuk urutan *job* A-B-C-D adalah sebagai berikut:

Langkah 1. Tempatkan *job* A di mesin M1 lalu hitung waktu penyelesaiannya (*Mspan1*).

Langkah 2. Lanjutkan pengerjaan *job* A di mesin M2 dan tempatkan *job* berikutnya (*job* B) di mesin M1 yang sedang tidak terpakai. Setelah itu, hitung total waktu penyelesaian *job* A dan *job* B di mesin M1 (*Mspan2*).

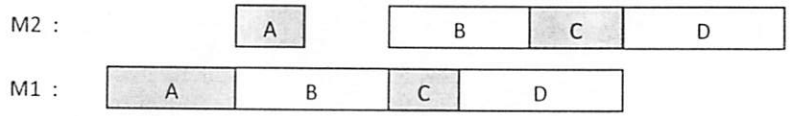


Langkah 3. Bandingkan *Mspan1* dan *Mspan2*. Apabila *Mspan1* lebih besar dibandingkan dengan *Mspan2*, penghitungan total *makespan* dimulai dari *Mspan1*. Apabila *Mspan1* lebih kecil dibandingkan dengan *Mspan2*, penghitungan total *makespan* dimulai dari *Mspan2*. Jika *Mspan1* sama dengan *Mspan2*, penghitungan total *makespan* boleh dimulai pada salah satu di antaranya.



Langkah 4. Ulangi Langkah 1 sampai Langkah 3 untuk *job* C dan *job* D.

Setelah keempat langkah di ats selesai dilakukan hingga *job* D, akan diperoleh *Gantt Chart* sebagai berikut:



Total *makespan* untuk urutan jadwal A-B-C-D di atas adalah:

$Z_p = t_{A(M1)} + t_{B(M1)} + t_{B(M2)} + t_{C(M2)} + t_{D(M2)} = 10 + 14 + 16 + 12 = 60$ satuan waktu

Nilai *fitness* kromosom dengan struktur ABCD adalah :

$$\Phi_p = \frac{1}{Z_p} = \frac{1}{60} = 0.016667$$

Tabel 5.2 Nilai *fitness* kromosom hasil hitungan enumeratif

No	Urutan Job	*Makespan*	*Fitness* = $\frac{1}{Makespan}$
1	ABCD	60	0.016667
2	ABDC	72	0.013889
3	ACBD	60	0.016667
4	ACDB	56	0.017857
5	ADBC	68	0.014706
6	ADCB	66	0.015152

No	Urutan Job	*Makespan*	*Fitness* = $\frac{1}{Makespan}$
7	BACD	60	0.016667
8	BADC	72	0.013889
9	BCDA	56	0.017857
10	BCAD	60	0.016667
11	BDAC	68	0.014706
12	BDCA	68	0.014706
13	CDAB	60	0.016667
14	CDBA	56	0.017857
15	CABD	60	0.016667
16	CADB	56	0.017857
17	CBDA	56	0.017857
18	CBAD	54	0.018519
19	DABC	68	0.014706
20	DACB	62	0.016129
21	DBAC	66	0.015152
22	DBCA	64	0.015625
23	DCAB	62	0.016129
24	DCBA	62	0.016129

Hasil Eksperimen Kasus 1

Hasil eksperimen untuk Kasus 1 disajikan dalam lima buah gambar grafik yaitu Gambar 5.3, Gambar 5.4, Gambar 5.5, Gambar 5.6, dan Gambar 5.7, serta satu buah tabel yaitu Tabel 5.3. Gambar 5.3 menunjukkan grafik nilai *fitness* tiap generasi selama 100 generasi pertama. Gambar 5.4 menunjukkan grafik frekuensi penyilangan dan mutasi tiap generasi untuk 100 generasi pertama. Gambar 5.5 menunjukkan grafik nilai *makespan* untuk 100 generasi pertama. Gambar 5.6 menunjukkan grafik nilai *fitness* maksimum, *fitness* rata-rata, dan *fitness* minimum tiap generasi, yang dimulai dari generasi ke-0

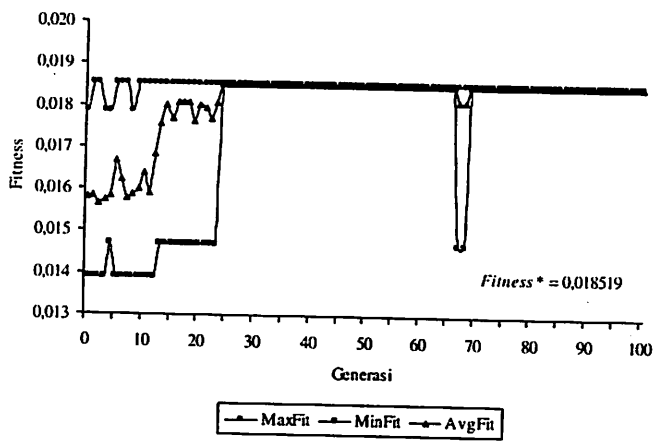
sampai generasi ke-500. Gambar 5.7 menunjukkan frekuensi penyilangan dan mutasi tiap generasi, dimulai dari generasi ke-0 sampai generasi ke-500. Tabel 5.3 memperlihatkan struktur-struktur kromosom terbaik yang pernah muncul dalam setiap generasi. Penjelasan masing-masing gambar di atas adalah sebagai berikut.

Gambar 5.3 memperlihatkan nilai *fitness* maksimum (*MaxFit*), nilai *fitness* minimum (*MinFit*), dan nilai *fitness* rata-rata (*AvgFit*) tiap generasi, dimulai dari generasi ke-0 hingga generasi ke-100. Untuk 25 generasi pertama, nilai *fitness* rata-rata menunjukkan peningkatan. Peningkatan ini menunjukkan membaiknya kinerja proses pencarian solusi optimum yang dilakukan oleh algoritma genetika. Semakin meningkat nilai *fitness* rata-rata, semakin membaik pula keadaan populasi. Kromosom-kromosom yang kurang baik (bernilai *fitness* kecil) banyak yang punah (*die-off*). Sebaliknya, kromosom-kromosom baru yang lebih baik (bernilai *fitness* lebih besar) mulai bermunculan. Munculnya kromosom-kromosom baru ini adalah akibat proses penyilangan, bukan mutasi. Kenyataan ini dapat dilihat pada Gambar 5.4 yang membuktikan bahwa selama 25 generasi pertama belum terjadi mutasi. Operator mutasi baru bekerja pada generasi ke-67.

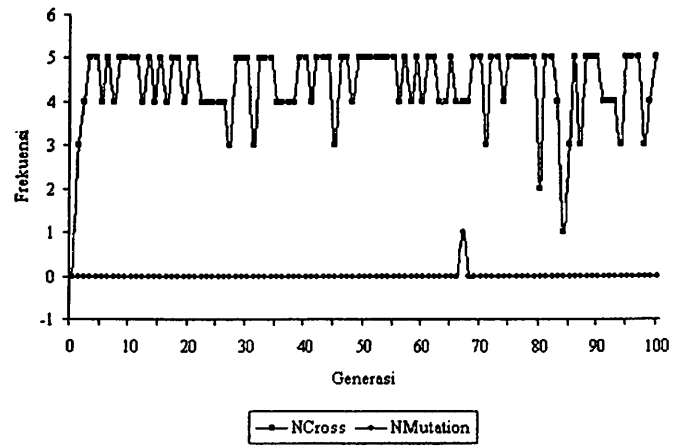
Seperti halnya nilai *fitness* rata-rata, nilai *fitness* minimum selama 25 generasi pertama juga menunjukkan peningkatan. Peningkatan ini disebabkan oleh hasil penyilangan yang mencetak kromosom-kromosom yang lebih baik dari kedua induknya. Peningkatan nilai *fitness* minimum akan berhenti pada nilai 0,018519 yang dimulai pada generasi ke-24. Setelah melewati generasi ke-24, nilai *fitness* minimum akan konvergen.

Sementara itu, nilai *fitness* maksimum menunjukkan sedikit perbedaan dalam hal kenaikan dibandingkan dengan nilai *fitness* maksimum dan nilai *fitness* rata-rata. Nilai *fitness* maksimum tertinggi yang pernah dihasilkan selama 100 generasi pertama adalah 0,018519

dan mulai konstan pada generasi ke-9 hingga generasi ke-100. Sebenarnya, nilai *fitness* sebesar 0,018519 sudah pernah dihasilkan pertama kali pada generasi ke-1, yang menandakan bahwa kromosom terbaik sudah pernah dihasilkan. Pada generasi ke-1 ini, jumlah kromosom terbaik ini hanya 10% dari total populasi yang ada. Pada generasi ke-3, kromosom terbaik ini punah (*die-off*) untuk sementara waktu akibat tidak terpilih menjadi kromosom induk. Pada generasi-generasi berikutnya kromosom terbaik ini muncul kembali dan jumlahnya dalam populasi mulai meningkat yang ditandai dengan mengarahnya nilai *fitness* maksimum, minimum, dan rata-rata ke suatu titik konvergensi. Titik konvergensi ini dimulai pada generasi ke-24.



Gambar 5.3 Grafik nilai *fitness* maksimum, *fitness* rata-rata, dan *fitness* minimum untuk 100 generasi pertama pada Kasus 1. (*PopSize=10; MaxGen=500; Pc=0,90; Pm=0,001*)



Gambar 5.4 Grafik frekuensi penyilangan dan mutasi untuk 100 generasi pertama pada Kasus 1. (*PopSize=10; MaxGen=500; Pc=0,90; Pm=0,001*)

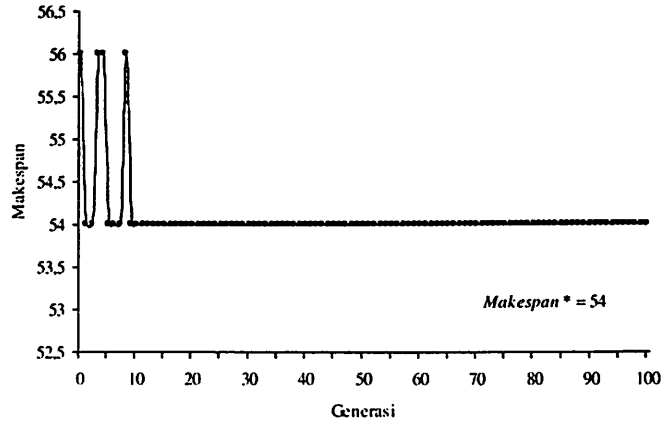
Nilai *fitness* rata-rata, *fitness* maksimum, dan *fitness* minimum mulai konvergen pada generasi ke-24 dengan nilai 0,018519. Hal ini menunjukkan bahwa keadaan populasi pada generasi ke-24 sudah homogen dalam hal kebugaran kromosomnya. Selain itu, berdasarkan hasil identifikasi terhadap struktur kromosom pada setiap generasi, diketahui bahwa populasi kromosom pada generasi ke-24 juga sudah homogen dalam hal struktur kromosomnya yaitu CDBA. Hasil identifikasi tersebut disajikan pada Tabel 5.4 yang memuat informasi struktur kromosom-kromosom terbaik yang pernah muncul pada setiap generasi. Kehomogenan nilai *fitness* akan bertahan hingga generasi ke-100, kecuali sempat turun pada generasi ke-67 yang diakibatkan oleh mutasi. Setelah generasi ke-67, populasi membaik kembali ke keadaan semula yaitu homogen dalam hal kebugaran dan struktur kromosom. Sampai generasi ke-100, proses pencarian yang dilakukan algoritma genetika telah melewati titik konvergensi. Struktur kromosom terbaik yang pernah ditemukan pada saat konvergensi maupun setelah titik

konvergensi adalah CDBA dengan nilai *fitness* 0,018519 dan nilai *makespan* sebesar 54 satuan waktu. Nilai *fitness* 0,018519 adalah nilai *fitness* terbaik yang pernah ditemukan selama 100 generasi yang dimiliki oleh kromosom dengan struktur CDBA.

Tabel 5.3 Daftar kromosom terbaik yang pernah dihasilkan dalam setiap generasi pada kasus 1

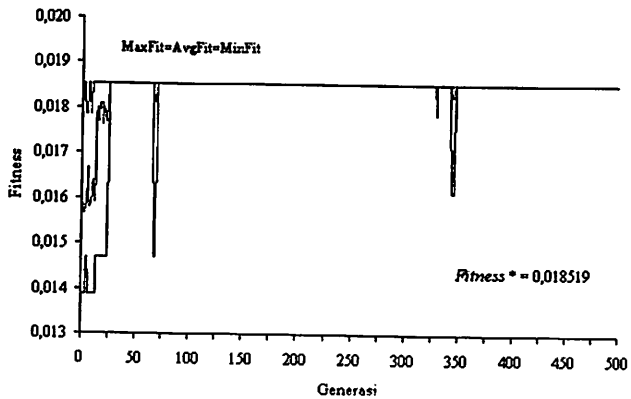
Generasi	Struktur Kromosom	*Fitness*	*Makespan*
0	CDAB	0,017857	56
	CBDA	0,017857	56
1	CDBA	0,018519	54
2	CDBA	0,018519	54
3	CBDA	0,017857	56
	CADB	0,017857	56
4	CBDA	0,017857	56
5	CDBA	0,018519	54
6	CDBA	0,018519	54
7	CDBA	0,018519	54
8	CDAB	0,017857	56
	CBDA	0,017857	56
9 — 500	CDBA	0,018519	54

Gambar 5.5 menunjukkan grafik nilai *makespan* terbaik (terkecil) yang pernah dihasilkan dalam setiap generasi. Nilai *makespan* terbaik ini diturunkan langsung dari nilai *fitness* terbaik, yaitu berupa nilai kebalikannya. Sebagai contoh, pada generasi ke-1, nilai *fitness* terbaik adalah 0,018519 sehingga nilai *makespan* terbaik untuk generasi ke-1 adalah 54 satuan waktu.



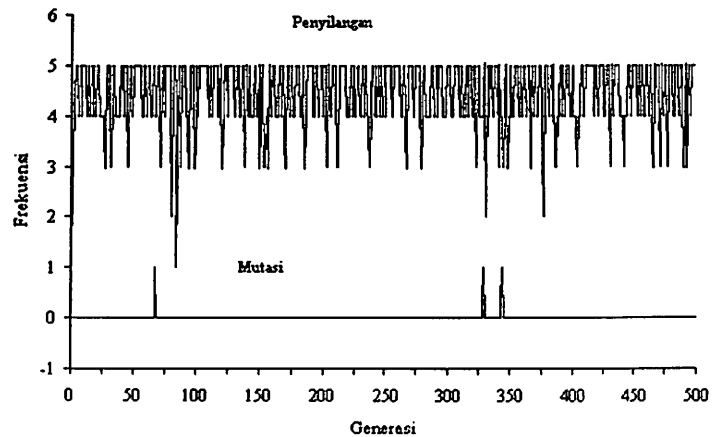
Gambar 5.5 Grafik nilai *makespan* terbaik yang pernah dicapai pada 100 generasi pertama untuk Kasus 1. (*PopSize=10; MaxGen=500; Pc=0,90; Pm=0,001*)

Jumlah generasi maksimum yang dipakai sebagai kriteria penghentian *running* program pada Kasus 1 adalah 500 generasi. Gambar-gambar yang telah disajikan sebelumnya hanya menunjukkan kinerja algoritma genetika hingga generasi ke-100. Untuk mengetahui bagaimana kinerja algoritma genetika setelah melewati generasi ke-100 hingga generasi ke-500, berikut ini disajikan hasil-hasil penelitian dalam dua buah gambar grafik yaitu Gambar 5.6 dan Gambar 5.7



Gambar 5.6 Grafik nilai *fitness* maksimum, *fitness* rata-rata, dan *fitness* minimum tiap generasi pada kasus 1 (*PopSize=10 ; MaxGen=500; Pc=0,90 ; Pm=0,001*)

Gambar 5.6 menunjukkan grafik nilai *fitness* maksimum, *fitness* rata-rata, dan *fitness* minimum tiap generasi yang dimulai dari generasi ke-0 hingga generasi ke-500. Gambar 5.6 ini merupakan kelanjutan dari Gambar 5.3. Dari Gambar 5.6 tersebut dapat diketahui bahwa antara generasi ke-325 hingga generasi ke-350 telah terjadi dua kali penurunan nilai *fitness* maksimum, nilai *fitness* rata-rata, maupun nilai *fitness* minimum populasi. Penurunan ini disebabkan oleh adanya mutasi, seperti yang terlihat pada grafik frekuensi penyilangan dan mutasi Gambar 5.7.



Gambar 5.7 Grafik frekuensi penyilangan dan mutasi tiap generasi pada kasus 1 (*PopSize=10 ; MaxGen=500; Pc=0,90 ; Pm=0,001*)

Penurunan nilai *fitness* antara generasi ke-325 dengan generasi ke-350 yang disebabkan oleh mutasi tidak mengubah status nilai *fitness* terbaik yang pernah dicapai sebelumnya. Setelah mutasi, populasi akan pulih kembali ke keadaan semula yaitu populasi yang homogen dalam hal kebugaran kromosomnya maupun struktur kromosomnya. Nilai *fitness* terbaik tetap 0,018519 yang dimiliki oleh kromosom dengan struktur CDBA. Nilai *fitness* ini akan bertahan hingga generasi ke-500.

Dari Gambar 5.6 dapat pula diketahui pula bahwa perbaikan nilai *fitness* yang dilakukan oleh algoritma genetika terjadi di awal-awal generasi, yaitu selama 25 generasi pertama. Untuk Kasus 1 ini, perbaikan nilai *fitness* selama 25 generasi pertama hanya dilakukan oleh operator penyilangan dengan frekuensi rata-rata 3–4 kali penyilangan dalam tiap generasi.

Efisiensi Algoritma Genetika

Seperti yang telah disebutkan di atas bahwa nilai *makespan* terkecil pada Kasus 1 pertama kali dicapai pada generasi ke-1. Sampai generasi ini, jumlah calon solusi yang telah dievaluasi (*Ns*) adalah:

Ns = (2 generasi) x (10 calon solusi/generasi) = 20 calon solusi

Jumlah total calon solusi dalam ruang pencarian (*Nt*) adalah:

Nt = (jumlah permutasi *job*) = 4! = 24 calon solusi

Persentase pencarian calon solusi yang dilakukan oleh algoritma genetika dalam *search space* (P_{Search}) adalah :

$$P_{Search} = (Ns / Nt) \times 100\% = (20 / 24) \times 100\% = 83{,}33\%$$

Besarnya persentase pencarian di atas menunjukkan bahwa algoritma genetika tidak cocok untuk memecahkan masalah *flow-shop* berskala kecil. Sebaliknya, algoritma genetika cocok untuk memecahkan masalah *flow-shop* berskala besar, seperti yang akan dibahas pada Kasus 2. Pada Kasus 2 tersebut, algoritma genetika hanya perlu mengeksplorasi 1,09% ruang pencarian untuk dapat menemukan solusi optimum.

5.6.2 Kasus 2: 8 *job* – 3 mesin

Kasus 2 (8 *job*-3 mesin) adalah pengembangan masalah Kasus 1 (4 *job*-2 mesin). Kasus 2 memiliki ruang pencarian yang lebih besar dibandingkan Kasus 1. Urutan pemrosesan yang mungkin ada 40.320 alternatif (8! = 40.320). Dengan demikian, besarnya ruang pencarian adalah 40.320 unit jadwal—yang direpresentasikan 40.320 unit kromosom.

Contoh Kasus 2 yaitu terdapat pada industri pakan ternak (McEllhiney 1994) yang memroses 8 produk pakan yang berbeda pada 3 buah mesin utama (mesin penggiling bahan baku, mesin pencampur,

dan mesin pemeletan). Setiap produk diproses dengan urutan yang sama. Tujuan penjadwalannya adalah menentukan urutan pakan ternak yang diproses sehingga dapat meminimumkan total waktu penyelesaian proses. Waktu pemrosesan *job* di setiap mesin untuk Kasus 2 ini dapat dilihat pada Tabel 5.4.

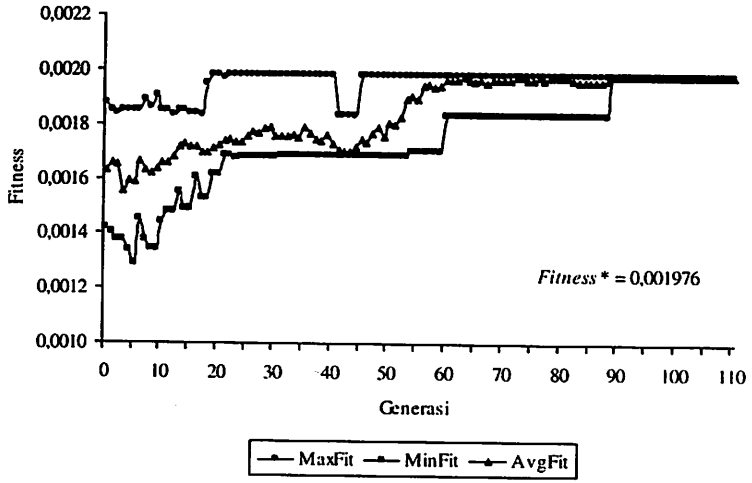
Tabel 5.4 Waktu pemrosesan *job* di setiap mesin untuk kasus 2

Mesin \ Job	A	B	C	D	E	F	G	H
M1	48	26	44	2	30	40	14	66
M2	70	56	6	74	62	96	52	78
M3	18	104	22	36	100	10	82	34

Hasil Eksperimen Kasus 2

Hasil eksperimen untuk Kasus 2 disajikan dalam 5 (lima) buah gambar grafik yaitu Gambar 5.8, Gambar 5.9, Gambar 5.10, Gambar 5.11, dan Gambar 5.12, serta satu buah tabel yaitu Tabel 5.5. Gambar 5.8 menunjukkan grafik nilai *fitness* tiap generasi selama 110 generasi pertama. Gambar 5.9 menunjukkan grafik frekuensi penyilangan dan mutasi tiap generasi untuk 110 generasi pertama. Gambar 5.10 menunjukkan grafik nilai *makespan* untuk 110 generasi pertama.

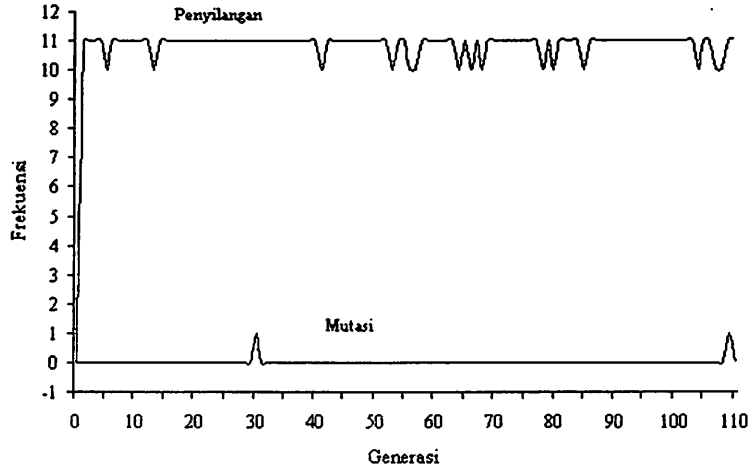
Gambar 5.11 menunjukkan grafik nilai *fitness* maksimum, *fitness* rata-rata, dan *fitness* minimum tiap generasi , yang dimulai dari generasi ke-0 sampai generasi ke-100. Gambar 5.12 menunjukkan frekuensi penyilangan dan mutasi tiap generasi, dimulai dari generasi ke-0 sampai generasi ke-1.000. Tabel 5.5 memperlihatkan struktur-struktur kromosom terbaik yang pernah muncul dalam setiap generasi. Penjelasan masing-masing gambar adalah sebagai berikut.



Gambar 5.8 Grafik nilai *fitness* maksimum, *fitness* rata-rata, dan *fitness* minimum untuk 110 generasi pertama pada kasus 2 (*PopSize=22; MaxGen=1000; Pc=0,99; Pm=0,001*)

Gambar 5.8 memperlihatkan nilai *fitness* maksimum (*MaxFit*), nilai *fitness* minimum (*MinFit*), dan nilai *fitness* rata-rata (*AvgFit*) tiap generasi, dimulai dari generasi ke-0 hingga generasi ke-110. Untuk 90 generasi pertama, nilai *fitness* rata-rata menunjukkan peningkatan. Peningkatan ini menunjukkan membaiknya kinerja proses pencarian solusi optimum yang dilakukan oleh algoritma genetika. Semakin meningkat nilai *fitness* rata-rata, semakin membaik pula keadaan populasi. Kromosom-kromosom yang kurang baik banyak yang punah. Sebaliknya, kromosom-kromosom baru yang lebih baik mulai bermunculan. Munculnya kromosom-kromosom baru ini adalah akibat proses penyilangan, bukan mutasi. Kenyataan ini dapat dilihat pada Gambar 5.9 yang membuktikan bahwa selama 90 generasi pertama, frekuensi kerja operator penyilangan sangat tinggi dibandingkan operator mutasi yang baru hanya bekerja terjadi satu kali, yaitu pada

generasi ke-30. Peningkatan nilai *fitness* rata-rata akan berhenti dan konstan pada nilai 0,001976 yang dimulai pada generasi ke-89. Seteleh melewati generasi ke-89, nilai *fitness* rata-rata akan konvergen.



Gambar 5.9 Grafik frekuensi penyilangan dan mutasi untuk 110 generasi pertama pada kasus 2 (*PopSize=22; MaxGen=1000; Pc=0,99; Pm=0,001*)

Seperti halnya nilai *fitness* rata-rata, nilai *fitness* minimum selama 90 generasi pertama juga menunjukkan peningkatan. Peningkatan ini disebabkan oleh hasil penyilangan yang berhasil mencetak kromosom-kromosom yang lebih baik dari kedua induknya. Peningkatan nilai *fitness* minimum akan berhenti pada nilai 0,001976 yang dimulai pada generasi ke-89. Setelah melewati generasi ke-89, nilai *fitness* minimum akan konvergen.

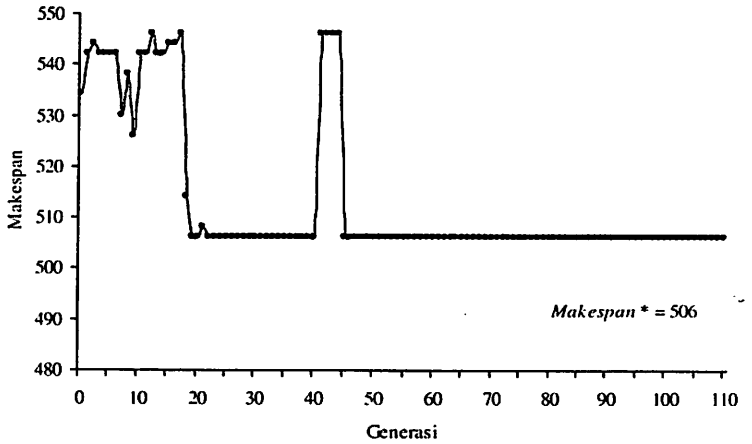
Berbeda dengan nilai *fitness* rata-rata dan *fitness* minimum, nilai *fitness* maksimum menunjukkan sedikit perbedaan dalam hal kenaikan nilai. Nilai *fitness* maksimum tertinggi yang pernah muncul selama 110 generasi pertama adalah 0,001976 dan mulai konstan pada generasi

ke-46. Sebenarnya, nilai *fitness* sebesar 0,001976 sudah pernah muncul pertama kali pada generasi ke-19. Hal ini menandakan bahwa kromosom terbaik (*the best chromosome*) sudah pernah dihasilkan. Tetapi kemudian pada generasi berikutnya, kromosom terbaik ini punah untuk sementara waktu. Kepunahan ini disebabkan oleh operator penyilangan, bukan mutasi. Operator mutasi yang terjadi pada generasi ke-30 sama sekali tidak mengubah nilai *fitness* maksimum yang sudah ada dalam populasi.

Nilai *fitness* rata-rata, *fitness* maksimum, dan *fitness* minimum mulai konvergen pada generasi ke-89 dengan nilai 0,0019769. Hal ini menunjukkan bahwa keadaan populasi pada generasi ke-89 sudah homogen dalam hal kebugaran kromosomnya (fenotipnya). Selain itu, berdasarkan hasil identifikasi terhadap struktur kromosom pada setiap generasi, diketahui bahwa populasi kromosom pada generasi ke-89 juga sudah homogen dalam hal struktur kromosomnya (genotipnya) yaitu DBEAGCHF. Hasil identifikasi tersebut disajikan pada Tabel 5.6 yang memuat informasi struktur kromosom-kromosom terbaik yang pernah muncul pada setiap generasi.

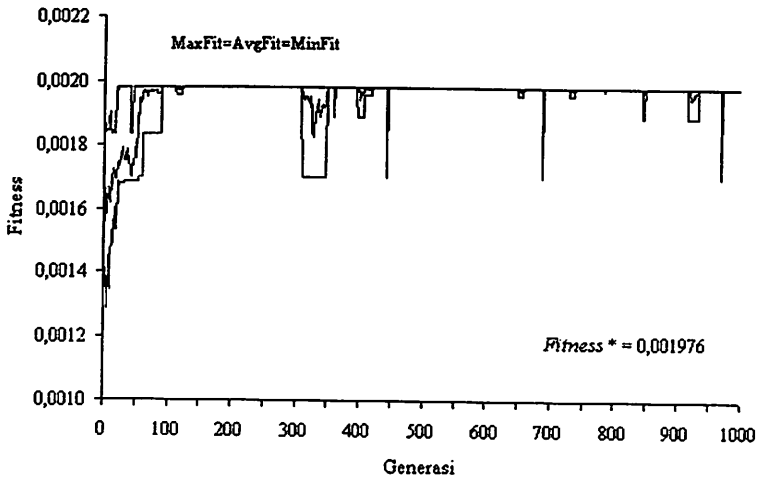
Kehomogenan nilai *fitness* akan bertahan hingga generasi ke-110. Pada generasi ke-109 terjadi keragaman struktur kromosom yaitu ada dua struktur kromosom berbeda yang memiliki nilai *fitness* sama besar. Keragaman ini muncul akibat adanya mutasi, seperti yang terlihat pada Gambar 5.8. Dua struktur kromosom tersebut adalah DBEAGCHF dan DBEGACHE. Setelah generasi ke-109, struktur kromosom populasi homogen kembali seperti semula yaitu DBEAGCHF. Sampai generasi ke-110, proses pencarian yang dilakukan algoritma genetika telah melewati titik konvergensi. Struktur kromosom terbaik yang pernah ditemukan pada saat konvergensi maupun setelah titik konvergensi adalah DBEAGCHF dan DBEGACHE dengan nilai *fitness* 0,001976 dan nilai *makespan* sebesar 506 satuan waktu.

Gambar 5.10 menunjukkan grafik nilai *makespan* terbaik (terkecil) yang pernah ditemukan dalam setiap generasi. Nilai *makespan* terbaik ini diturunkan langsung dari nilai *fitness* terbaik, yaitu berupa nilai kebalikannya. Sebagai contoh, pada generasi ke-89, nilai *fitness* terbaik adalah 0,001976 sehingga nilai *makespan* terbaik untuk generasi ke-89 adalah 506 satuan waktu.



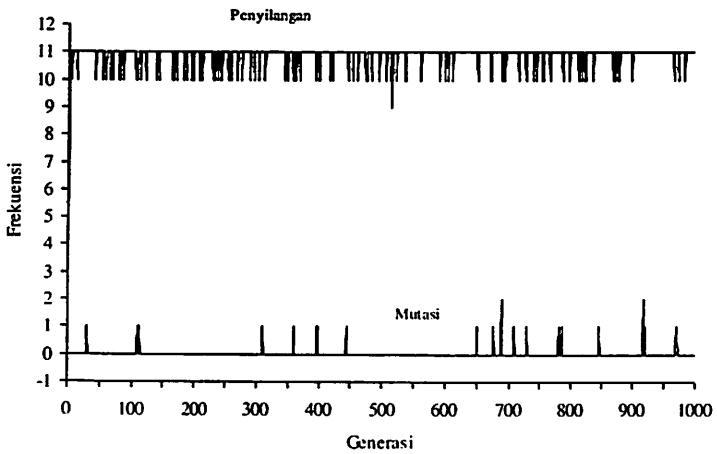
Gambar 5.10 Grafik nilai *makespan* terbaik yang pernah dicapai pada 110 generasi pertama untuk kasus 2 (*PopSize=22; MaxGen=1000; Pc=0,99; Pm=0,001*)

Jumlah generasi maksimum yang dipakai sebagai kriteria penghentian *running* program pada Kasus 2 adalah 1.000 generasi. Gambar-gambar yang telah disajikan sebelumnya hanya menunjukkan kinerja algoritma genetika hingga generasi ke-110. Untuk mengetahui bagaimana kinerja algoritma genetika setelah melewati generasi ke-110 hingga generasi ke-1.000, berikut ini disajikan hasil-hasil penelitian dalam dua buah gambar grafik yaitu Gambar 5.11 dan Gambar 5.12.



Gambar 5.11 Grafik nilai *fitness* maksimum, *fitness* rata-rata, dan *fitness* minimum tiap generasi pada kasus 2 (*PopSize=22; MaxGen=1000; Pc=0,99; Pm=0,001*)

Gambar 5.11 menunjukkan grafik *fitness* maksimum, *fitness* rata-rata, dan *fitness* minimum tiap generasi yang dimulai dari generasi ke-0 hingga generasi ke-1.000. Gambar 5.11 ini merupakan kelanjutan dari Gambar 5.8. Dari Gambar 5.11, dapat diketahui bahwa antara generasi ke-110 sampai generasi ke-1.000 telah terjadi beberapa kali penurunan nilai *fitness* maksimum, nilai *fitness* rata-rata, maupun nilai *fitness* minimum populasi. Penurunan ini disebabkan oleh adanya mutasi, seperti yang terlihat pada grafik frekuensi penyilangan dan mutasi Gambar 5.12.



Gambar 5.12 Grafik frekuensi penyilangan dan mutasi tiap generasi pada kasus 2 (*PopSize=22; MaxGen=1000; Pc=0,99; Pm=0,001*)

Penurunan nilai *fitness* yang terjadi antara generasi ke-110 dengan generasi ke-1.000 yang disebabkan oleh mutasi tidak mengubah status nilai *fitness* terbaik yang pernah dicapai sebelumnya. Nilai *fitness* terbaik tetap 0,001976 yang bertahan hingga generasi ke-1.000.

Selama 1.000 generasi, nilai *fitness* terbaik memang tidak berubah. Tetapi, selama 1.000 generasi telah dihasilkan beberapa struktur kromosom berbeda yang memiliki nilai *fitness* sama besar. Jika selama 110 generasi pertama hanya ada dua struktur kromosom terbaik (DBEAGCHF dan DBEGACHE) yang memiliki nilai *fitness* sebesar 0,001976, setelah 1.000 generasi ada tiga struktur kromosom baru terbaik yang dihasilkan yang sama-sama memiliki nilai *fitness* 0,001976. Ketiga kromosom tersebut adalah DBEACGHF, FBEAGCHD, dan DBEAGHCF. Informasi detail tentang kromosom terbaik yang pernah ditemukan dalam setiap generasi dapat dilihat pada Tabel 5.5.

Tabel 5.5 Daftar kromosom terbaik yang pernah dihasilkan dalam setiap generasi pada kasus 2

Generasi	Struktur Kromosom	*Fitness*	*Makespan*
0	DBE AFC GH	0,001873	534
1	ABH CGF DA	0,001845	542
2	GEF BAH CD	0,001838	544
3	EGH CBF DA	0,001845	542
4	EGH CBF DA	0,001845	542
5	EBH CGD FA	0,001845	542
6	EBH DGC FA	0,001845	542
	EBH CGD FA	0,001845	542
7	DBE AGF CH	0,001887	530
8	BGE FHD CA	0,001859	538
9	GBE FHD CA	0,001901	526
10	EBH DGF CA	0,001845	542
11	EGH DBF CA	0,001845	542
12	GAE FBC DH	0,001832	546
13	GAE CBF DH	0,001845	542
14	GBE CAF DH	0,001845	542
15	GBE CAF HD	0,001838	544
	GFE CAB HD	0,001838	544
16	GBE CAF HD	0,001838	544
17	DBE AGF HC	0,001832	546
18	DBE CGF HA	0,001946	514
19	DGE ABC HF	0,001976	506
	DCE AGB HF	0,001976	506
20	DBG AEC HF	0,001976	506
21	DEG ABC HF	0,001976	506
22	DBG AEC HF	0,001976	506
23	DBE AGC HF	0,001976	506
24	DBE AGC HF	0,001976	506
	DBG AEC HF	0,001976	506
25–28	DBE AGC HF	0,001976	506
29	DBE AGC HF	0,001976	506
	DBC AGE HF	0,001976	506
30	DBE AGC HF	0,001976	506
	DBC AGE HF	0,001976	506
31	DBE AGC HF	0,001976	506
	DBC AGE HF	0,001976	506
32–40	DBE AGC HF	0,001976	506
41–44	DBE AGF HC	0,001976	546
45–108	DBE AGC HF	0,001976	506

Generasi	Struktur Kromosom	*Fitness*	*Makespan*
109	DBE AGC HF	0,001976	506
	DBE GAC HF ($)	0,001976	506
110–675	DBE AGC HF	0,001976	506
676	DBE AGC HF	0,001976	506
	DBE ACG HF ($)	0,001976	506
677–687	DBE AGC HF	0,001976	506
688	DBE AGC HF	0,001976	506
	DBE ACG HF ($)	0,001976	506
	FBE AGC HD ($)	0,001976	506
689–707	DBE AGC HF	0,001976	506
708	DBE AGC HF	0,001976	506
	DBE AGH CF ($)	0,001976	506
709–779	DBE AGC HF	0,001976	506
780	DBE AGC HF	0,001976	506
	DBE GAC HF ($)	0,001976	506
781–916	DBE AGC HF	0,001976	506
917	DBE AGC HF	0,001976	506
	DBE ACG HF ($)	0,001976	506
918	DBE AGC HF	0,001976	506
919	DBE AGC HF	0,001976	506
920	DBE AGC HF	0,001976	506
	DBE ACG HF ($)	0,001976	506
921–970	DBE AGC HF	0,001976	506
971	DBE AGC HF	0,001976	506
	DBE GAC HF	0,001976	506
972–1000	DBE AGC HF	0,001976	506

Keterangan : $ = kromosom yang mengalami mutasi

Efisiensi Algoritma Genetika

Seperti yang telah disebutkan di atas bahwa nilai *makespan* terkecil pada Kasus 2 pertama kali dicapai pada generasi ke-19. Sampai generasi ini, jumlah calon solusi yang telah dievaluasi (*Ns*) oleh algoritma genetika mencapai:

$Ns =$ (20 generasi) x (22 calon solusi/generasi) = 440 calon solusi

Jumlah total calon solusi dalam ruang pencarian (*Nt*) adalah:

$$Nt = (\text{jumlah permutasi } job) = 8! = 40320 \text{ calon solusi}$$

Persentase pencarian calon solusi yang dilakukan oleh algoritma genetika dalam ruang pencarian (P_{Search}) adalah:

$$P_{Search} = (Ns / Nt) \times 100\% = (440 / 40320) \times 100\% = 1{,}09\ \%$$

Persentase di atas membuktikan bahwa pencarian solusi optimum (kromosom terbaik) yang dilakukan oleh algoritma genetika sangat efisien. Di mana nilai *makespan* terkecil yang ditemukan oleh algoritma genetika hanya mengekplorasi 1,09% ruang pencarian.

5.7 Kesimpulan

Masalah penjadwalan *flow-shop* berskala besar dikenal sebagai masalah optimasi yang rumit yang tidak dapat dipecahkan dengan menggunakan teknik-teknik optimasi konvensional, seperti *calculus-based technique* atau *enumerative search*. Alasan yang paling mendasar adalah teknik-teknik tersebut sangat bergantung pada adanya informasi tambahan berupa fungsi turunan, memerlukan waktu komputasi yang lama, serta mudah terjebak ke dalam perolehan nilai optimum lokal.

Algoritma genetika, sebagai teknik optimasi yang dipakai dalam penelitian ini, telah terbukti sangat efisien untuk memecahkan masalah *flow-shop* berskala besar. Pada dua kasus yang digunakan, algoritma genetika terbukti sebagai teknik pencarian yang sangat efisien. Untuk Kasus 2 yaitu masalah penjadwalan 8 *job*-3 mesin, algoritma genetika hanya perlu mengeksplorasi 1,09% ruang pencarian untuk mendapatkan solusi optimal.

Berbeda dengan teknik-teknik optimasi konvensional yang hanya memberikan satu solusi pemecahan masalah, algoritma genetika dapat

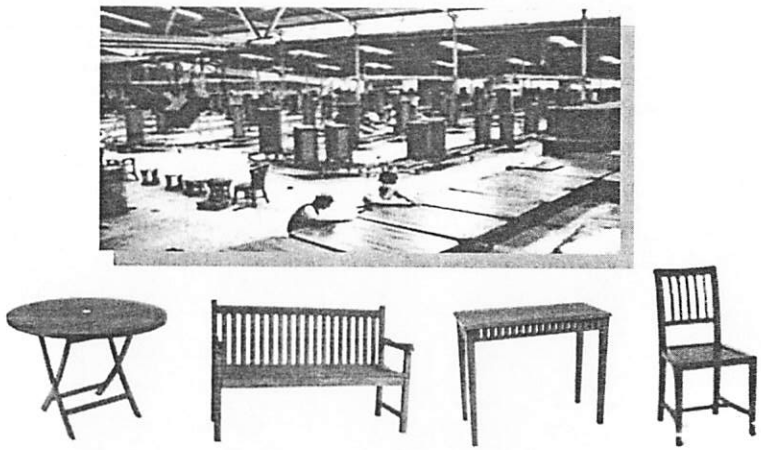
menawarkan beberapa solusi (kromosom terbaik) sekaligus yang memiliki nilai fungsi tujuan (nilai *fitness*) yang sama. Untuk Kasus 2, alternatif solusi penjadwalan yang diperoleh adalah D-B-E-A-G-C-H-F, D-B-E-G-A-C-H-E, D-B-E-A-C-G-H-F, F-B-E-A-G-C-H-D, dan D-B-E-A-G-H-C-F. Nilai *makespan* untuk kelima jadwal di atas adalah 506 satuan waktu dan nilai *fitness*-nya adalah 0,001976. Sedangkan untuk Kasus 1, masalah penjadwalan 4 *job*-2 mesin, solusi pejadwalan yang diperoleh hanya ada satu yaitu C-D-B-A. Nilai *makespan* untuk jadwal tersebut adalah 54 satuan waktu dan nilai *fitness*-nya adalah 0,018519.

Bab 6
Aplikasi Algoritma Genetika pada Optimasi Penjadwalan Produksi Meubel Kayu

Penjadwalan produksi meubel kayu merupakan aktivitas yang cukup rumit dan kompleks karena harus mendayagunakan sumber daya manusia, peralatan, dan waktu secara efisien agar target produksi dapat tercapai sesuai dengan pemesanan yang ada. Problema yang dihadapi dalam produksi meubel kayu ini adalah ketidakpastian jadwal kerja sehingga mengitroduksi adanya kondisi di mana peralatan tersedia, tetapi operator (pekerja) tidak tersedia (waktu *idle* mesin) atau sebaliknya operator tersedia, tetapi peralatan tidak tersedia (waktu idle operator) sehingga penyelesaian pemesanan meubel kayu terlambat. Dengan banyaknya pesanan yang diterima perusahan produksi meubel kayu, maka penjadwalan produksi harus dilakukan dengan optimal dengan cara memperkecil terjadinya waktu yang terbuang (*idle*) baik pada pekerja maupun peralatan yang terlibat. Melakukan optimasi penjadwalan dalam hal ini adalah mencari semua kemungkinan kombinasi penjadwalan yang terbaik yang dapat meminimisasi waktu *idle* dalam produksi meubel kayu. Pencarian kombinasi penjadwalan optimum dari semua ruang solusi (*solution space*) penjadwalan yang mungkin adalah pekerjaan yang dapat diselesaikan dengan algoritma genetika.

6.1 Pendekatan Masalah Penjadwalan dan Representasi Masalah Formulasi

Dalam contoh kasus produksi meubel kayu (Sidharta 2005), beberapa hal yang perlu diperhitungkan antara lain jenis dan jumlah produk, jenis dan jumlah mesin (peralatan) produksi, serta jenis dan jumlah pekerjaan (*job*). Contoh tipe (jenis) produksi meubel kayu adalah meja, bangku, kursi seperti disajikan pada Gambar 6.1. Untuk satu tipe produk meubel kayu dapat melibatkan beberapa pekerjaan (*job*), dan setiap *job* bisa terdiri atas beberapa tahap yang melibatkan mesin tertentu dalam perioda (waktu) eksekusi tertentu pula. Contoh deskripsi job dan tahapan yang diperlukan serta jenis dan jumlah mesin yang digunakan disajikan pada Tabel 6.1 dan 6.2. Tabel 6.3 merepresentasikan keterkaitan antara jenis (tipe) produk, *job,* dan tahapan yang diperlukan, serta mesin dan waktu eksekusi yang diperlukan.



Gambar 6.1 Produksi dan jenis produk meubel kayu (Sidharta 2005)

Sebagai ilustrasi dari Tabel 6.3, jenis produk meubel tipe *2210* memerlukan 4 *job* (*job 1, job 2, job 3, job 4*) di mana *job 1* memerlukan 7 tahap (*tahap 1* s/d *tahap 7*). Untuk *job 1* pada *tahap 1* diperlukan *mesin 1* dengan waktu eksekusi *5 unit waktu* (dinotasikan sebagai *1,5*); untuk *job 1* pada *tahap 2* diperlukan *mesin 2* dengan waktu eksekusi *6 unit waktu* (dinotasikan sebagai *2,6*); untuk *job 3* pada *tahap 4* diperlukan *mesin 12* dengan waktu eksekusi 3 *unit waktu.* Demikian seterusnya representasi dan interpretasi untuk tipe produk meubel kayu yang lain pada Tabel 6.3.

Tabel 6.1 Contoh deskripsi sebagian tipe produk dan *job* yang diperlukan untuk produksi meubel kayu

Tipe Produk	Job	Keterangan
2210	Job 1	Pembuatan top table
	Job 2	Pembuatan penyangga top table
	Job 3	Pembuatan kaki meja
	Job 4	Pembuatan sambungan kaki meja
2211	Job 1	Pembuatan top table
	Job 2	Pembuatan pinggiran top table
	Job 3	Pembuatan penyangga top table
	Job 4	Pembuatan kaki meja
	Job 5	Pembuatan sambungan kaki meja
2212	Job 1	Pembuatan top table
	Job 2	Pembuatan pinggiran top table
	Job 3	Pembuatan penyangga top table
	Job 4	Pembuatan kaki meja
	Job 5	Pembuatan sambungan kaki meja
2213	Job 1	Pembuatan top table
	Job 2	Pembuatan kaki meja
	Job 3	Pembuatan sambungan kaki meja
2214	Job 1	Pembuatan top table
	Job 2	Pembuatan penyangga top table
	Job 3	Pembuatan kaki meja
	Job 4	Pembuatan sambungan kaki meja

Tabel 6.2 Contoh deskripsi sebagian jenis, fungsi, dan jumlah unit mesin produksi meubel kayu

No	Nama Alat Produksi	Fungsi	Jumlah
1	*Band Saw*	Membelah kayu gelondongan menjadi papan	4 unit
2	*Rip Saw*	Memotong papan (vertikal)	4 unit
3	*Cross-cut Saw*	Memotong papan (horisontal)	2 unit
4	*Thicknessing Planner*	Pembuatan komponen kecil	2 unit
5	*Tennoner*	Pembuatan komponen sambungan kayu	4 unit
6	*Morticer*	Pembuatan komponen sambungan kayu	4 unit
7	*Spidle Moulder*	Pembuatan komponen kecil	2 unit
8	*Router*	Pembuatan komponen kecil	4 unit
9	*Lathe*	Pembuatan komponen kecil	2 unit
10	*Auto-shaping Machine*	Pembuatan komponen kecil	3 unit
11	*Circular Saw*	Pembuatan komponen kecil	2 unit
12	*Radial Arm Saw*	Pembuatan komponen kecil	2 unit
13	*Cut-off Saw*	Pembuatan komponen kecil	2 unit
14	*Laminating Flat Press*	Pelaminatingan pelapis kayu	3 unit
15	*Frame Press*	Mesin *press*	2 unit
16	Bubut	Pembuatan komponen kecil	8 unit
17	*Sander*	Penghalusan kayu	8 unit

6.2 Representasi Kromosom

Berdasarkan representasi tabular dari Tabel 6.3 tersebut, representasi kromosom penjadwalan menggunakan *string* dengan *permutation encoding* di mana satu gen merepresentasikan urutan dan nomor *job* serta nomor *tahap* job. Selanjutnya setiap gen akan dipetakan oleh suatu fungsi yang menghasilkan nomor mesin yang digunakan dan waktu eksekusi pekerjaan yg diperlukan. Salah satu representasi string kromosom tipe produk *2210* disajikan pada Gambar 6.2 yang merepresentasikan urutan pekerjaan dari semua kemungkinan permutasi dari *string* kromosom yang bisa dibangkitkan.

Tabel 6.3 Representasi sebagian jenis produk, *job*, tahap serta mesin dan waktu eksekusi

Tipe Produk	Job	Tahap 1	Tahap 2	Tahap 3	Tahap 4	Tahap 5	Tahap 6	Tahap 7
2210	Job 1	1,5	2,6	5,3	6,4	11,2	14,3	17,6
	Job 2	1,5	2,4	3,6	5,2	7,6	10,5	17,5
	Job 3	2,3	5,5	6,5	12,6	15,3	17,4	
	Job 4	3,3	5,5	8,4	16,5	17,4		
2211	Job 1	1,5	2,5	5,2	6,4	11,4	14,3	17,6
	Job 2	1,5	2,5	3,6	5,4	7,6	10,5	17,5
	Job 3	2,4	5,6	6,5	12,6	15,2	17,5	
	Job 4	2,5	6,6	13,4	17,6			
	Job 5	3,6	8,3	5,3	9,2	11,4	14,5	17,4
2212	Job 1	1,5	2,4	8,2	9,2	13,6	14,3	17,5
	Job 2	3,5	5,5	8,4	16,3	7,6	10,4	17,5
	Job 3	2,4	5,6	6,2	8,4	16,5	17,3	
	Job 4	2,5	5,5	6,5	17,6			
	Job 5	3,6	8,3	5,3	9,2	11,4	14,5	
2213	Job 1	1,5	2,4	5,5	8,4	16,6	14,3	17,6
	Job 2	3,2	13,6	8,2	9,2	7,6	10,6	17,5
	Job 3	2,4	5,6	6,5	8,4	16,2	17,5	
2214	Job 1	1,3	2,5	5,3	6,2	7,2	17,4	
	Job 2	1,4	2,5	7,4	10,3	12,3	17,3	
	Job 3	8,4	9,4	11,2	13,2	15,3		
	Job 4	1,4	5,3	6,4	15,5	17,3		

1-1	3-3	2-3	4-3	2-2	4-1	3-1	1-3	4-2	2-1	3-2	1-2

Gambar 6.2 Contoh representasi *string* urutan pekerjaan untuk produksi tipe produk *2210*

Pada Gambar 6.2, gen pertama yang berisi 1-1 merepresentasikan urutan pertama pekerjaan produksi pada *job 1* pada *tahap 1*; gen kedua yang berisi 3-3 merepresentasikan urutan kedua pekerjaan produksi pada *job 3* pada *tahap 3;* gen ketiga yang berisi 2-3 merepresentasikan urutan ketiga pekerjaan produksi pada *job 2* pada *tahap 3;* dan

seterusnya hingga gen ke-12 (terakhir) yang berisi 1-2 merepresentasikan urutan keduabelas pekerjaan produksi pada *job 1* pada *tahap 2*. Selanjutnya gen 1-1 akan diterjemahkan oleh suatu fungsi yang memetakan nilai 1-1 *(job 1 tahap 1)* ke suatu nilai 1,5 (lihat Tabel 6.3) yang merepresentasikan *nomor mesin (jenis mesin)* dan *unit waktu* eksekusinya, yaitu *mesin 1* dengan *5 unit waktu* eksekusi.

6.3 Penggunaan Operator Genetika

Pada kasus penjadwalan produksi meubel ini digunakan tiga jenis operator yaitu *seleksi, crossover,* dan *mutasi*. Operator seleksi yang digunakan adalah *elitism* di mana *string* terbaik pada tiap generasi akan otomatis diturunkan ke generasi berikutnya (Koza 2001). Operator *crossover* yang digunakan adalah *precedence preservative crossover (PPX)* di mana *string* baru disusun secara acak dari gen dua *string* induk. Angka acak 1 dan 2 digunakan untuk memilih induk. Jika angka acak 1, diturunkan gen paling kiri dari induk pertama lalu gen tersebut dihapus dari induk kedua; jika angka acak 2, diturunkan gen paling kiri dari induk kedua lalu gen tersebut dihapus dari induk pertama. Proses tersebut dilakukan sampai semua karakter dalam *string* kedua induk tersebut habis. Sebagai contoh untuk dua *string* induk *ABCDEF* dan *CABFDE* dan angka acak *121122* akan menghasilkan *string* baru *ACBDFE*. Operator *mutasi* memlih satu gen secara acak dan karakter di posisi tersebut dihapus, kemudian satu gen dipilih, dan gen yang telah dihapus tadi disisipkan. Sebagai contoh *string ABCDEF string ABCDEF* yang dimutasikan pada gen ke-3 dan ke-5 menghasilkan *string* baru *ABDECF*.

6.4 Penetapan Fungsi *Fitness*

Pada kasus penjadwalan produksi meubel ini, fungsi *fitness* untuk menentukan keberlanjutan suatu *string* pada suatu populasi didasarkan pada jumlah waktu efektif pengoperasian mesin pada jadwal produksi. Dengan demikian, faktor yang memengaruhi nilai *fitness* penjadwalan adalah prosedur konversi *string* menjadi sebuah jadwal dengan memperhitungkan waktu eksekusi dan waktu tunda (*delay/idle*) pada sebuah jadwal. Selanjutnya fungsi *fitness* diformulasikan sebagai fungsi efektivitas waktu penggunaan mesin sebagai berikut. Makin tinggi nilainya, fungsi *fitness* makin baik.

$$f(x) = \frac{t_o}{t_o - t_d}$$

Di mana: x : *string*, t_o: waktu penggunaan mesin, t_d: waktu tunda mesin.

6.5 Penetapan Parameter Genetika

Parameter genetika berguna untuk pengendalian operator-operator genetika yang digunakan sehingga kinerja algoritma genetika dapat dioptimalkan. Parameter yang digunakan adalah ukuran populasi, jumlah generasi, probabilitas *crossover* (Pc), dan probabilitas *mutasi* (P_m). Pada ukuran populasi yang kecil hanya tersedia sedikit pilihan untuk *crossover* dan sebagian kecil saja *string* yang dieksplorasi untuk setiap generasinya. Pada ukuran populasi yang besar akan melibatkan jumlah iterasi yang besar pula karena jumlah domain solusi yag dieksplorasi semakin besar. Hal ini akan mengintroduksi komputasi yang lebih banyak sehingga kinerja algoritma genetika akan menurun. Merujuk pada Bodenhofer (2003) dan Kusumadewi (2003), ukuran populasi yang disarankan antara 20–30, probabilitas *crossover* berkisar 80–95%, dan probabilitas *mutasi* berkisar 0,5–1%. Untuk itu pada pen-

jadwalan ini ditetapkan ukuran populasi sebesar 20, jumlah generasi sebesar 100, probabilitas *crossover* 80%, dan probabilitas *mutasi* 1%.

6.6 Pemodelan Matematika Penjadwalan

Penjadwalan yang dimaksud adalah proses penyusunan jadwal dari *string*. Untuk menjelaskan algoritma penjadwalan produksi meubel kayu dalam kasus yang dibahas ini perlu ditetapkan beberapa notasi seperti tercantum pada Tabel 6.4.

Tabel 6.4 Notasi dalam pemodelan matematika penjadwalan

Notasi	Penjelasan
O	Operasi (*job*)
O_n	Operasi ke *n*, di mana *1<n<jumlah operasi pada string*
$Ts(O_n)$	Waktu *start* dari O_n
$Tp(O_n)$	Perioda eksekusi mesin oleh O_n
$Tf(O_n)$	Waktu *finish* dari O_n
M^i	Nomor mesin jenis *i*

Merujuk pada notasi di Tabel 7.4, algoritma transformasi *string* ke jadwal dikembangkan sebagai berikut.

Algoritma 1: Transformasi *string* ke jadwal

1. Mulai
2. Tetapkan A sebagai himpunan semua operasi O
3. Gabungkan semua operasi di A yang mempunyai nomor urut (order) terkecil dan masukkan dalam himpunan B, sehingga $B \subset A$, $B = \{ O \in A | O$ dengan nomor order terkecil$\}$
4. Gabungkan semua operasi di B yang mempunyai nomor tahap produksi terkecil dan masukkan dalam himpunan C sehingga $C \subset B$, $C = \{ O \in B | O$ dengan nomor tahap produksi terkecil$\}$

5. Untuk semua $O \in C$
   i. Jika O_n adalah operasi yang letaknya paling kiri pada *string*, konfirmasi keberadaan operasi yang mendahului O_n pada *himpunan* A:
      - Jika tidak ada operasi sebelumnya yang mendahului O_n, $Ts(O_n) = 0$.
      - Jika ada operasi sebelumnya (O_{n-1}) yang mendahului O_n, $Ts(O_n) = Ts(O_{n-1}) + Tp(O_{n-1})$

   ii. Jika M^i adalah mesin yg digunakan O_n maka
      a. Untuk setiap D di mana $D = \{M_k | M_k$ adalah mesin yang bertipe sama dengan M^i $1 \leq k \leq$ jumlah mesin yang bertipe sama dengan $M^i\}$, konfirmasi operasi sebelumnya pada setiap M_k:
         - Jika tidak ada operasi sebelumnya pada M_k, O_n ditugaskan pada M_k dan dimasukkan ke jadwal, $D = \emptyset$ dan $O_n \notin A$.
         - Jika ada operasi sebelumnya pada M_k, hitung $Tf_{n-1}(M_k)$, tetapkan $Tf_{n-1}(M_k) \in E$ dan $M_k \notin D$, ulangi langkah (5.ii.a) hingga $D = \emptyset$
      b. Untuk semua $Tf_{n-1}(M_k) \in E$.
         Bandingkan setiap nilai $Tf_{n-1}(M_k)$ dengan $Ts(O_n)$.
         - Jika $Tf_{n-1}(M_k) < Ts(O_n)$ tetapkan bahwa $Tf_{n-1}(M_k) \notin E$.
         - Selainnya, jika $Tf_{n-1}(M_k) = Ts(O_n)$ maka O_n masuk ke jadwal untuk diproses pada M_k yang memiliki nilai $Tf_{n-1}(M_k) = Ts(O_n)$, tetapkan $E = \emptyset$ dan $O_n \notin A$.
         - Selainnya, jika $Tf_{n-1}(M_k) > Ts(O_n)$ maka tetapkan $Tf_{n-1}(M_k) \in F$ dan $Tf_{n-1}(M_k) \notin E$, ulangi langkah (5.ii.b) hingga $E = \emptyset$.

c. Untuk semua $Tf_{n-1}(M_k) \in F$.

Bandingkan nilai $Tf_{n-1}(M_k)$ dengan semua $\in F$.

- Jika $Tf_{n-1}(M_k)^1 = Tf_{n-1}(M_k)^2$, di mana $Tf_{n-1}(M_k)^1$ & $Tf_{n-1}(M_k)^2$ anggota ke-1 dan ke-2 dari F, *tetapkan* $Tf_{n-1}(M_k)^2 \notin F$, tetapkan O_n masuk ke jadwal untuk diproses pada M_k yang memiliki nilai $Tf_{n-1}(M_k) \in F$, tetapkan $F=\varnothing$ dan $O_n \notin A$.
- Selainnya, jika $Tf_{n-1}(M_k)^1 < Tf_{n-1}(M_k)^2$, tetapkan $Tf_{n-1}(M_k)^2 \notin F$, ulangi langkah (5.ii.c) sampai $F = \{ Tf_{n-i}(M_k)^i \}$, tetapkan O_n masuk ke jadwal untuk diproses pada M_k yang memiliki nilai $Tf_{n-1}(M_k) \in F$, tetapkan $F=\varnothing$ dan $O_n \notin A$.
- Selainnya, jika $Tf_{n-1}(M_k)^1 > Tf_{n-1}(M_k)^2$, tetapkan $Tf_{n-1}(M_k)^1 \notin F$, ulangi langkah (5.ii.c) sampai $F = \{ Tf_{n-1}(M_k)^2 \}$, tetapkan O_n masuk ke jadwal untuk diproses pada M_k yang memiliki nilai $Tf_{n-1}(M_k) \in F$, tetapkan $F=\varnothing$ dan $O_n \notin A$.

6. Ulangi langkah (5) sampai $C=\varnothing$
7. Ulangi langkah (4) sampai $C=\varnothing$
8. Ulangi langkah (3) sampai $C=\varnothing$
9. Selesai

Algoritma 2: Optimasi Penjadwalan

Proses optimasi penjadwalan dengan algoritma genetika diformulasikan sebagai berikut:

1. Mulai
2. Tetapkan jumlah populasi (P_{max}), probabilitas *crossover* (P_c), probabilitas *mutasi* (P_m), *banyaknya* elitism (*nElit*), jumlah generasi maksimal (*MaxGen*).
3. Baca *nomor urut* x, *tipe produk* y, dan *jumlah pesanan* z.

4. Duplikasikan tiap *record input* sebanyak *n* kali sehingga untuk tiap record *z=10*.
5. Untuk tiap record yang terbentuk, berikan *job* dan *tahap* produksi yang harus dilalui dengan nilai waktu eksekusi mesin *p X z*
6. Bentuk populasi awal secara acak (dari Tabel 6.3) sebanyak P_{max} *string*, susun jadwal dengan ***algoritma 1***, hitung *fitness string* $f(x)$, untuk setiap *string* (x)
7. Bentuk populasi berikutnya:
   - 7.1 Proses seleksi: masukkan *nElit* buah *string* terbaik ke populasi berikut.
   - 7.2 Proses *crossover:* dengan *Roulette Wheel Selection*, pilih 2 *string* induk lalu lakukan crossover, ulangi sampai sampai jumlah populasi mencapai $P_c X P_{max}$.
   - 7.3 Proses mutasi: dengan *Roulette Wheel Selection* pilih sebanyak $P_{max} - (P_c X P_{max})$ *string* untuk dimutasi.
   - 7.4 Susun jadwal tiap *string* baru dengan **algoritma 1**, hitung *fitness string* $f(x)$, untuk setiap *string* (x).
8. Ulangi langkah (5) sampai *MaxGen* tercapai.
9. Hasil akhir adalah jadwal yang dibentuk *string* dengan nilai *fitness* terbaik (tertinggi).
10. Selesai.

6.7 Implementasi Penjadwalan dengan Algoritma Genetika

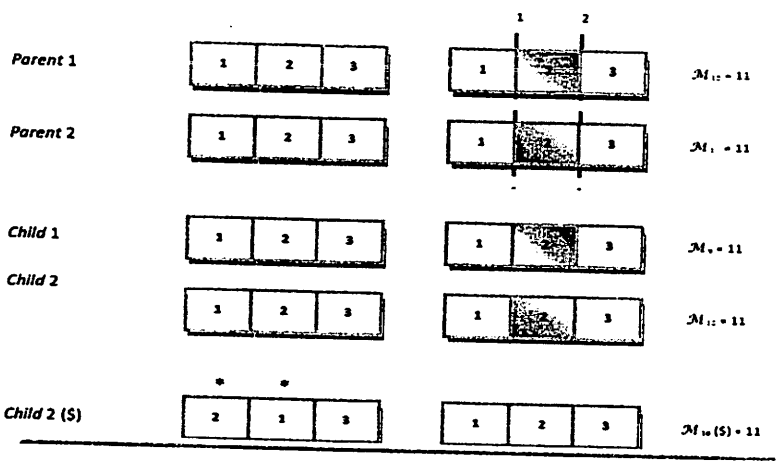
Implementasi protipe sistem menggunakan *Microsoft Visual Basic 6.0* pada lingkungan sistem operasi *Windows XP*. Gambar 7.3 adalah tampilan *interface* sistem penjadwalan ketika dimasukan pesanan nomor pemesanan ke-1 dengan tipe produk *2211* sebanyak *5.000* unit;

	<p1,p2>	<Mach.Cross>	<Csite1>	<Csite2>	<Mach.Mutate>	Job		Makespan	
1)	(9,17)	2	1	2	0	213	213	11	
2)	(9,17)	2	1	2	0	123	213	11	
3)	(12, 7)	2	1	2	0	213	123	11	
4)	(12, 7)	2	1	2	0	123	123	11	
5)	(5, 6)	2	1	2	0	123	123	11	
6)	(5, 6)	2	1	2	0	123	123	11	
7)	(18,15)	2	1	2	0	123	123	11	
8)	(18,15)	2	1	2	0	123	123	11	
9)	(10, 1)	2	1	2	0	123	123	11	
10)	(10, 1)	2	1	2	0	123	123	11	
11)	(9, 5)	2	1	2	1	213	123	11	$
12)	(9, 5)	2	1	2	0	213	123	11	
13)	(3, 3)	2	1	2	0	123	213	11	
14)	(3, 3)	2	0	0	0	123	213	11	
15)	(20,16)	2	0	0	0	123	213	11	
16)	(20,16)	1	1	2	0	123	123	11	
17)	(3, 9)	1	1	2	0	213	123	11	
18)	(3, 9)	2	1	2	0	123	213	11	
19)	(17,10)	2	1	2	0	213	213	11	
20)	(17,10)	2	1	2	0	123	123	11	

SumFitness value of generation = 220.0000
Max. Fitness value of generation = 11.0000
Min. Fitness value of generation = 11.0000
Avg. Fitness value of generation = 11.0000
Cum. of NCross until gen. 7 = 60
Cum. of NMutation until gen. 7 = 1

Gambar 7.6 Populasi P(7) terjadi mutasi pada kasus 1

Proses mutasi tersebut dapat dilihat pada Gambar 7.7.



Gambar 7.7 Mutasi kromosom(10) pada generasi ke-7 untuk kasus 1

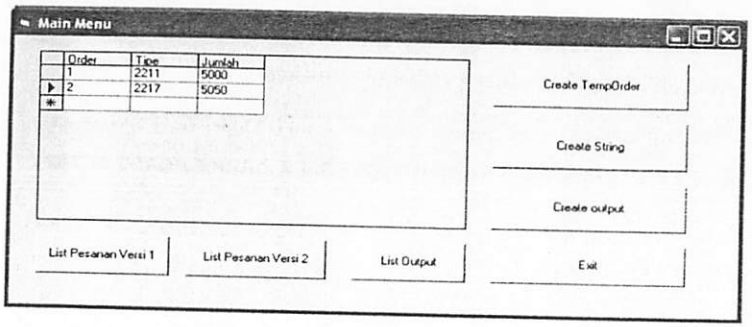
Mutasi terjadi pada kromosom(10), generasi ke-7. Subkromosom (*Mach.Mutate*) yang terpilih adalah subkromosom ke-1, yaitu [1 2 3] dan setelah mengalami mutasi maka gen 1 dan 2 saling bertukaran. Kromosom baru hasil proses mutasi adalah [2 1 3] [1 2 3] yang memiliki nilai *makespan* yang sama dengan nilai *makespan* kromosom induknya. Nilai *makespan* tetap 11.

Seluruh proses pada sistem ***GA_JobShop*** mulai dari inisialisasi populasi awal, evaluasi kromosom, seleksi kromosom, proses penyilangan, serta proses mutasi merupakan tahapan-tahapan yang menjadi rangkaian pencarian dalam menemukan solusi optimum. Pencarian akan dihentikan apabila kriteria penghentian terpenuhi. Kriteria penghentian yang digunakan pada sistem ***GA_JobShop*** adalah jumlah generasi maksimum (*MaxGen*). Untuk Kasus 1 generasi maksimumnya adalah sebanyak 100 generasi, artinya proses pencarian akan berhenti setelah mencapai generasi ke-100.

e. Hasil Running Sistem GA_JobShop

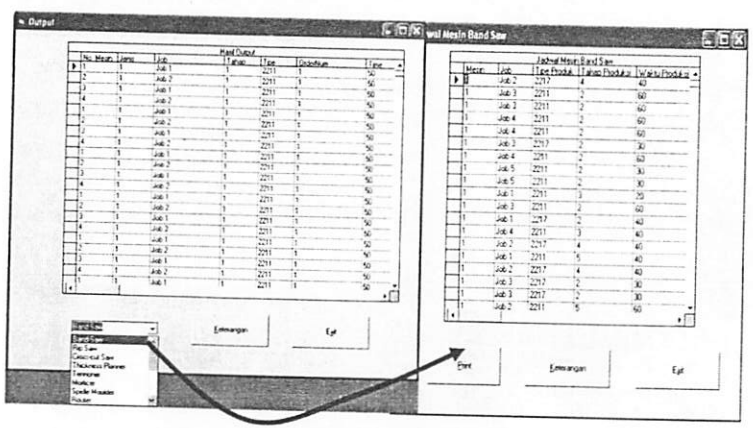
Hasil *running* sistem *GA_JobShop* untuk Kasus 1 diilustrasikan dalam dua buah grafik. Grafik yang pertama yaitu terdapat pada Gambar 7.8, menunjukkan nilai *makespan* maksimum (*Max*), *makespan* rata-rata (*Avg*), dan *makespan* minimum (*Min*) dari generasi awal (generasi ke-0) hingga generasi maksimum (generasi ke-100). Grafik berikutnya terdapat pada Gambar 7.9, menunjukkan frekuensi terjadinya penyilangan dan mutasi sepanjang 100 generasi.

sedangkan pemesanan ke-2 dengan tipe produk *2217* sebanyak *5.050* unit.



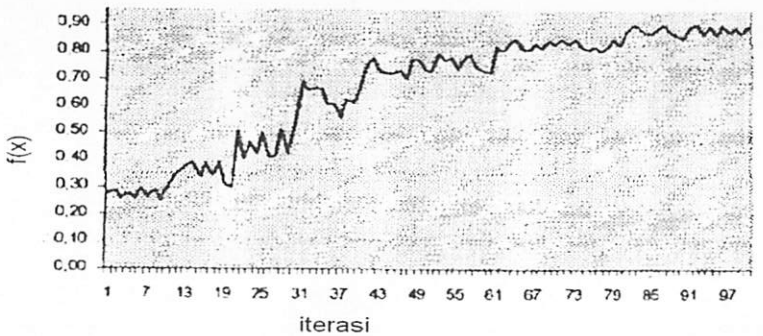
Gambar 6.3 Contoh tampilan pemesanan produk.

Setelah diproses sistem penjadwalan optimalnya, diperoleh luaran (*ouput*) penggunaan tiap mesin untuk *job* dan tahapannya seperti disajikan pada Gambar 6.4. Untuk melihat jadwal penggunaan mesin tertentu, misalnya *band saw*, tinggal klik pada pilihan *item* mesin tersebut di sebelah kiri bawah dan akan tampil jadwal mesin tersebut di window sebelah kanan.



Gambar 6.4. Contoh luaran penjadwalan penggunaan mesin untuk tiap *job* dan tahapannya

Jadwal yang dihasilkan dari penjelasan di atas adalah jadwal optimal dengan nilai fungsi *fitnes* tertinggi. Berikut hasil iterasi algoritma genetika yang ditampilkan dalam bentuk grafik (Gambar 6.5) yang menunjukkan kenaikan nilai fungsi *fitness* pada setiap iterasi.



Gambar 6.5 Grafik nilai fungsi *fitness* yang semakin meningkat dari pada tiap iterasi dari contoh kasus yang diberikan (Sidharta 2005)

Bab 7
Aplikasi Algoritma Genetika dalam Sistem Penjadwalan Bidang Agroindustri

Bab 7 akan membahas mengenai aplikasi algoritma genetika lainnya yaitu sistem penjadwalan produksi. Sistem ini telah diteliti oleh Ayip Bayu Satrio dan Yandra Arkeman (2007) di mana kegunaannya untuk menyusun jadwal mesin produksi pada perusahaan manufaktur agroindustri.

7.1 Latar Belakang

Penjadwalan produksi di dalam dunia industri, baik industri manufaktur maupun agroindustri, memiliki peranan penting sebagai bentuk pengambilan keputusan. Perusahaan berupaya untuk memiliki penjadwalan yang paling efekif dan efisien sehingga dapat meningkatkan produktivitas yang dihasilkan dengan total biaya dan waktu seminimum mungkin.

Dalam sebuah sistem produksi yang kompleks dapat terjadi penumpukan pekerjaan atau barang yang membentuk antrian panjang yang tidak dapat diselesaikan secara optimal. Sistem produksi yang melibatkan banyak proses, mesin, dan juga waktu proses yang bervariasi akan menemui banyak hambatan bila tidak ada metode penjadwalan yang tepat dan akhirnya berakibat pada proses produksi secara keseluruhan. Sistem tidak dapat bekerja secara efektif dan efisien.

Penjadwalan produksi melakukan pembebanan mesin berdasarkan rencana proses yang telah dibuat perencana proses. Permasalahan yang sering muncul adalah rencana proses dibuat tidak memerhatikan beban kerja mesin sehingga sering terjadi penumpukan pada mesin unggul. Solusi permasalahan dapat dilakukan dengan memerhatikan alternatif urutan proses.

Salah satu masalah dalam penjadwalan produksi adalah adanya kesulitan menemukan teknik yang paling tepat untuk membuat jadwal produksi yang paling baik, optimal, dan memenuhi segala kriteria-kriteria penjadwalan yang telah ditetapkan. Teknik-teknik penyusunan jadwal produksi yang sudah ada (teknik konvensional) tidak dapat dipakai karena teknik-teknik tersebut memiliki banyak kelemahan dalam menangani masalah berskala besar dan kompleks.

Permasalahan penjadwalan *job shop* (*job shop scheduling*) dengan fungsi tujuan meminimumkan total waktu proses (*makespan*) dapat diterjemahkan sebagai memproses setiap pekerjaan (*job*) dari **n** *job* pada **m** mesin dengan urutan tertentu. Setiap pekerjaan terdiri atas serangkaian operasi. Setiap mesin dapat menangani tidak lebih dari satu pekerjaan pada suatu waktu dan setiap pekerjaan mengunjungi mesin hanya satu kali.

Alternatif urutan proses meningkatkan fleksibilitas produksi karena alternatif solusi lebih banyak. Sebagai konsekuensi, waktu pencarian solusi menggunakan metode optimasi naik secara eksponensial. Pendekatan heuristik akan dilakukan untuk memecahkan permasalahan penjadwalan dengan memerhatikan alternatif urutan proses. Pendekatan heuristik yang digunakan adalah algoritma genetika. Pemilihan ini didasarkan bahwa algoritma penjadwalan yang ada tidak dapat melakukan penjadwalan sekaligus memilih alternatif urutan proses.

Untuk mendapatkan hasil penjadwalan yang optimum maka digunakan algoritma genetika. Sebuah solusi penjadwalan dikatakan optimum apabila memiliki waktu rata-rata produksi seminimum mungkin. Hasil operasi genetik akan diubah kembali menjadi sebuah jadwal yang mudah dipahami oleh pengguna.

Sistem ini akan dicari solusi untuk menyelesaikan permasalahan penjadwalan *job shop* dengan menggunakan metode *Genetic Algorithms* (Algoritma Genetika). Prinsip kerjanya yaitu berdasarkan analogi evolusi biologi, yang terdiri atas proses penginisialisasian populasi, proses penyeleksian, proses penyilangan, dan proses mutasi. Keunggulan dari algoritma genetika ini adalah strukturnya yang sederhana, mudah mengimplementasikannya, dan cepat dalam pencapaian solusi yang optimum (efektif dan efisien).

Penjadwalan produksi yang kurang baik dapat mengakibatkan keterlambatan pengiriman produk. Penjadwalan *job shop* merupakan suatu permasalahan optimasi kombinatorial yang kompleks yang membutuhkan waktu komputasi yang besar dalam proses pencarian solusi terbaiknya. Algoritma Genetika mampu menyelesaikan permasalahan penjadwalan *job shop* dengan waktu komputasi yang singkat.

7.2 Formulasi Masalah

Penjadwalan secara umum didefinisikan sebagai penetapan waktu dari penggunaan peralatan, fasilitas, dan aktivitas manusia dalam sebuah organisasi. Penjadwalan produksi terdiri atas tiga tahapan utama. Pertama adalah tahapan *loading*, yaitu penentuan *job* yang harus diproses untuk setiap mesin. Kedua adalah tahapan *sequencing*, yaitu penentuan urutan pengerjaan untuk masing-masing *job*. Ketiga adalah tahapan *detailed scheduling*, yaitu pengaturan waktu mulai dan selesainya suatu *job*.

Penjadwalan bagi perusahaan manufaktur adalah aspek yang sangat penting karena penjadwalan merupakan salah satu bagian perencanaan dan pengendalian produksi. Jadwal yang baik diperoleh dari algoritma yang baik pula, sehingga perlu untuk menentukan suatu algoritma yang tepat untuk suatu masalah penjadwalan.

Penjadwalan tipe *job shop* merupakan penjadwalan yang kompleks karena merupakan kombinasi pengurutan *job-job* pada tiap mesin yang memprosesnya. Sebuah industri dengan tipe sistem produksi *job-shop* menggunakan parameter optimasi yaitu nilai *makespan* (waktu penyelesaian pekerjaan secara keseluruhan). Semakin meningkatnya alternatif penjadwalan produksi secara eksponensial, dengan banyaknya sumber daya yang digunakan merupakan permasalahan pada optimasi kombinatorial yang kompleks. Kombinasi yang optimum akan menghasilkan nilai *makespan* yang minimum.

Permasalahan pada penjadwalan tipe *job shop* yang rumit dan kompleks membutuhkan suatu metode yang mampu memecahkan masalah tersebut yang tidak dapat dipecahkan oleh metode-metode konvensional. Penggunaan algoritma genetika, yaitu metode pencarian secara acak yang meniru proses alam (proses evolusi), dapat memecahkan masalah penjadwalan tersebut dengan solusi yang mendekati optimum namun hanya membutuhkan waktu dan usaha yang lebih kecil.

7.3 Pengembangan Sistem

Sistem ini dinamakan ***GA_JobShop*** dimana terdapat tiga kasus yang akan dipecahkan yaitu Kasus ke-1 merupakan kasus penjadwalan 3 *job*-2 mesin; Kasus ke-2 merupakan kasus penjadwalan 3 *job*- 3 mesin; Kasus ke-3 merupakan kasus penjadwalan 5 *job*-12 mesin. Kasus ke-1 merupakan skenario penjadwalan yang sangat sederhana (*numerical example*) dan Kasus ke-2 merupakan contoh penjadwalan

tipe *job shop* yang diambil dari literatur yang terdapat pada buku *Genetic Algorithm and Engineering Design* (Gen dan Cheng 1997). Kasus ke-3 merupakan studi kasus industri peralatan pengolahan hasil pertanian. Ketiga kasus tersebut dioptimasi untuk didapatkan nilai *makespan* yang minimum.

Data skenario pada Kasus 1 dirancang sebagai *numerical example*. Data tersebut dihitung nilai *makespan*-nya dengan cara enumeratif kemudian dibandingkan dengan hasil yang diperoleh dari pencarian algoritma genetika sebagai verifikasi dan validasi sistem yang telah dibuat. Verifikasi merupakan pemeriksaan apakah logika pada sistem sesuai dengan logika penjadwalan tipe *job shop*, sedangkan validasi merupakan proses penentuan apakah hasil *output* pada sistem merupakan representasi yang akurat dari aplikasi nyata (Hoover dan Perry 1989). Apabila hasil perhitungan dengan sistem sudah sama dengan hasil perhitungan enumeratif, sistem ini dapat digunakan untuk memecahkan masalah dengan skala yang lebih besar, seperti Kasus 2 dan Kasus 3.

7.3.1 Kasus 1: Penjadwalan *job shop* kasus 3 *job*-2 mesin

Masalah penjadwalan 3 *job*-2 mesin berarti masalah penempatan *job-job* pada mesin-mesin yang sesuai. Berdasarkan representasi kromosom jenis *preference list based representation*, mesin-mesin yang ada digolongkan menjadi subkromosom yang memuat daftar *job-job* di dalamnya, sehingga kasus 3 *job*-2 mesin memiliki kemungkinan sebanyak $(3!)^2 = 36$ calon solusi. Didapatkan persamaan umum untuk peluang representasi kromosom (*total search space*) adalah $(n!)^m$, dengan n adalah banyaknya *job* dan m adalah banyaknya mesin.

Sistem ***GA_JobShop*** akan diuji kebenarannya melalui data numerik pada Kasus 1 ini. Data ini dibuat dengan mengikuti aturan penjadwalan tipe *job shop*. Nilai *makespan* yang diperoleh dari hasil *output* sistem ***GA_JobShop*** akan dibandingkan dengan perhitungan secara enumeratif,

yaitu menghitung semua alternatif yang ada (*total search space*). Contoh kasusnya adalah sebagai berikut:

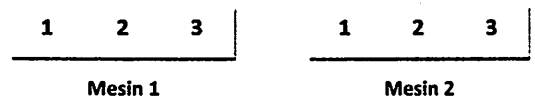
Tabel 7.1 Waktu proses, urutan mesin, dan urutan proses pada Kasus 1 (3 *job*-2 mesin)

Waktu Proses			Urutan Mesin		
Job	Urutan Proses		*Job*	Urutan Proses	
	1	2		1	2
j_1	2	3	j_1	m_1	m_2
j_2	2	3	j_2	m_2	m_1
j_3	5	4	j_3	m_1	m_2

Salah satu karakteristik penjadwalan tipe *job shop* adalah tiap-tiap *job* memiliki urutan proses yang berbeda-beda. Pada Tabel 7.1 dapat dijelaskan bahwa Kasus 1 memiliki tiga *job* yang harus diselesaikan pada dua mesin yang berbeda. *Job* 1 memiliki urutan proses yang berbeda dengan *job* 2. *Job* 1 harus melewati mesin 1 terlebih dahulu kemudian dilanjutkan dengan mesin 2. Mesin 2 menjadi mesin yang didahulukan oleh *job* 2. *Job* 3 memiliki urutan proses yang sama dengan *job* 1, yaitu mesin 1 setelah itu mesin 2.

Masing-masing mesin memiliki waktu proses yang berbeda untuk *job-job* yang berbeda. Mesin 1 memproses *job* 1 selama 2 satuan waktu, memproses *job* 2 selama 3 satuan waktu, dan memproses *job* 3 selama 5 satuan waktu. Sedangkan mesin 2 memproses *job* 1 selama 3 satuan waktu, memproses *job* 2 selama 2 satuan waktu, dan memproses *job* 3 selama 4 satuan waktu.

Tiap-tiap *job* yang melewati mesin-*m* akan menjadi suatu gen di dalam subkromosom (mesin-*m*). Representasi kromosomnya dapat dilihat pada Gambar berikut:



Gambar 7.1 Representasi kromosom penjadwalan tipe *job shop* kasus 3 *job*-2 mesin

a. Parameter-parameter Algoritma Genetika

Nilai-nilai parameter algoritma genetika yang digunakan untuk Kasus 1 adalah sebagai berikut:

- Peluang penyilangan (*Pc*) = 0,9
- Peluang mutasi (*Pm*) = 0,01
- Bilangan acak (0–1) = 0,6
- Jumlah populasi (*PopSize*) = 20
- Jumlah generasi maksimum = 100

b. Inisialisasi Populasi Awal

Populasi awal P(0) dibangkitkan secara acak oleh sistem dan hasil *running* dapat dilihat pada Gambar di bawah ini:

```
=====================================
          INITIAL REPORT
Simple Function Optimization with GAs
        JOB SHOP SCHEDULING
=====================================

GAs parameters
==============
Pop. Size              = 20
Chrom. Length          = 6
Max. Generation        = 100
Crossover probability     = 0.9000
Mutation probability      = 0.0100

INITIAL GENERATION STASTISTICS
==============================
        Job       Makespan
------------------------------
  1) 312  132         14
  2) 231  213         14
  3) 132  123         11
  4) 213  213         11
  5) 231  213         14
  6) 321  231         12
  7) 231  123         14
  8) 213  123         11
  9) 132  321         14
 10) 321  321         12
 11) 321  123         14
 12) 213  123         11
 13) 321  321         12
 14) 231  213         14
 15) 213  321         14
 16) 132  213         11
 17) 123  123         11
 18) 213  123         11
 19) 132  213         11
 20) 213  123         11
------------------------------

Sum of Fitness =    247.0000
Max. Fitness   =     14.0000
Min. Fitness   =     11.0000
Avg. Fitness   =     12.3500
```

Gambar 7.2 Populasi awal P(0) untuk kasus 1

Pada populasi ke-0 nilai *makespan* tertinggi yaitu sebesar 14 dan terendah sebesar 11. Nilai *makespan* rata-rata pada populasi awal tersebut yang telah dibangkitkan oleh bilangan acak sebesar 0,6 adalah 12,35.

c. Evaluasi dan Seleksi Kromosom

Setiap kromosom-*k* dalam populasi dievaluasi dengan menghitung nilai *fitness*-nya setelah terbentuk populasi awal. Pada

kasus penjadwalan tipe *job shop*, nilai *makespan* merupakan nilai *fitness*.

Setelah semua kromosom dievaluasi nilai *makespan*-nya, proses selanjutnya adalah proses seleksi kromosom. Sepasang kromosom akan dipilih secara acak untuk disilangkan membentuk sepasang kromosom baru. Teknik seleksi yang digunakan adalah teknik seleksi turnamen (*tournament selection*). Berikut adalah langkah-langkah teknik seleksi turnamen pada Kasus 1:

Langkah 1. Pilih dua buah kromosom secara acak dalam populasi

Langkah 2. Bandingkan nilai *makespan* kedua buah kromosom tersebut

Langkah 3. Kromosom akan terpilih apabila nilai *makespan* kromosom tersebut lebih kecil dari nilai *makespan* kromosom yang lain

Kromosom-kromosom yang terpilih sebagai kromosom induk dari P(0) adalah sebagai berikut:

Kromosom(2), *makespan*-nya → $M_2 = 14$

Kromosom(4), *makespan*-nya → $M_3 = 11$

Kromosom(6), *makespan*-nya → $M_5 = 12$

Kromosom(8), *makespan*-nya → $M_6 = 11$

Kromosom(9), *makespan*-nya → $M_7 = 14$

Kromosom(10), *makespan*-nya → $M_8 = 12$

Kromosom(11), *makespan*-nya → $M_9 = 14$

Kromosom(12), *makespan*-nya → $M_{13} = 11$

Kromosom(13), *makespan*-nya → $M_{14} = 12$

Kromosom(15), *makespan*-nya → $M_{15} = 14$

Kromosom(16), *makespan*-nya → $M_{17} = 11$

Kromosom(17), *makespan*-nya → $M_{18} = 11$

Kromosom(18), *makespan*-nya → $M_{19} = 11$

Kromosom(19), *makespan*-nya → $M_{18} = 11$

Kromosom(20), *makespan*-nya → $M_{19} = 11$

d. Penyilangan dan Mutasi

Proses penyilangan terjadi setelah proses seleksi. Dua buah (sepasang) kromosom induk diberikan suatu peluang untuk dapat melakukan melakukan proses penyilangan. Pada Kasus 1 ini peluang penyilangan (*Pc*) yang digunakan adalah 0,9. Nilai 0,9 berarti diharapkan 90% populasi yang terbentuk pada generasi selanjutnya adalah hasil penyilangan generasi sebelumnya. Teknik penyilangan yang digunakan adalah *Partially Mapped Crossover* (PMX) yang telah dimodifikasi pada penelitian ini. Berikut adalah tahapan-tahapan proses penyilangan:

Tahapan 1. Mengambil sepasang kromosom induk hasil proses seleksi.

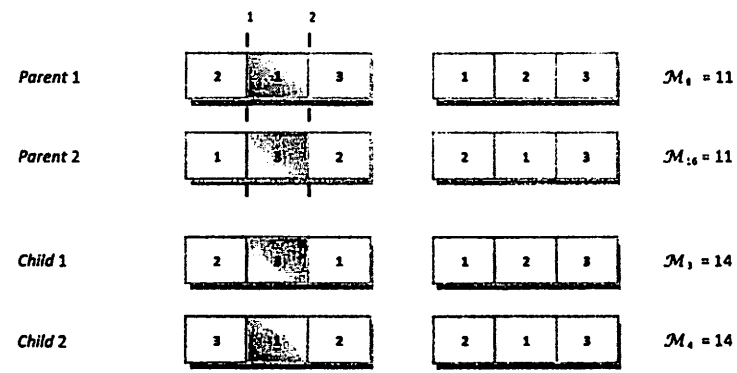
Tahapan 2. Menentukan salah satu subkromosom secara acak pada masing-masing kromosom induk yang akan disilangkan.

Tahapan 3. Menentukan dua buah titik penyilangan secara acak pada masing-masing subkromosom yang terpilih secara acak.

Tahapan 4. Menyilangkan dua buah subkromosom tersebut dengan aturan PMX.

Tahapan 5. Muncul dua buah kromosom baru hasil penyilangan dan masukkan ke dalam populasi berikutnya.

Dari populasi P(0) diambil sebagai contoh yang mengalami penyilangan adalah kromosom(8) dan kromosom(16). Subkromosom (*Mach.Cross*) yang terpilih adalah subkromosom ke-1 dengan titik penyilangannya adalah 1 dan 2. Sepasang kromosom yang disilangkan tersebut menghasilkan dua buah kromosom baru yang nilai *makespan*-nya lebih besar dari kedua induknya. Dua buah kromosom baru tersebut menjadi anggota populasi baru P(1) sebagai kromosom(1) dan kromosom(2). Ilustrasi proses penyilangan tersebut dapat dilihat pada Gambar 7.3.



Gambar 7.3 Proses penyilangan kromosom(8) dan kromosom(16) generasi ke-0 untuk kasus 1

Pada Gambar 2 dapat dilihat bahwa subkromosom ke-1 dari kromosom(8) adalah [2 1 3] dengan titik penyilangan 1 dan 2 maka *job* 1 terpilih sebagai gen yang mengalami proses penyilangan. Subkromosom ke-1 dari kromosom(16) adalah [1 3 2] dan gen yang mengalami penyilangan adalah gen 3. Setelah itu gen-gen yang terpilih yaitu 1 dan 3 saling bertukaran. Sesuai dengan metode PMX, hasil

penyilangan kromosom(8) dan kromosom(16) adalah [2 3 1] [1 2 3] dan [3 1 2] [2 1 3]. Kromosom(8) dan kromosom(16) memiliki nilai *makespan* sebesar 11, setelah mengalami proses penyilangan mengalami perubahan nilai *makespan*, yaitu terjadi peningkatan nilai menjadi 14 pada masing-masing kromosom anak.

Gambar 7.4 merupakan populasi baru P(1) yang terbentuk dari penyilangan kromosom-kromosom di populasi awal P(0).

	<p1,p2>	<Mach.Cross>	<xSite1>	<xSite2>	<Mach.Mutate>	Job	Makespan
1)	(4,12)	2	1	2	0	213 123	11
2)	(4,12)	2	1	2	0	213 213	11
3)	(8,16)	1	1	2	0	231 123	14
4)	(8,16)	1	1	2	0	312 213	14
5)	(8,13)	2	1	2	0	213 123	11
6)	(8,13)	2	1	2	0	321 321	12
7)	(15,20)	1	1	2	0	213 321	14
8)	(15,20)	1	1	2	0	213 123	11
9)	(17,18)	1	1	2	0	213 123	11
10)	(17,18)	1	1	2	0	123 123	11
11)	(6, 9)	1	0	0	0	321 231	12
12)	(6, 9)	1	0	0	0	132 321	14
13)	(11,18)	1	1	2	0	312 123	14
14)	(11,18)	1	1	2	0	123 123	11
15)	(2, 4)	2	1	2	0	231 213	14
16)	(2, 4)	2	1	2	0	213 213	11
17)	(10,19)	1	1	2	0	231 321	12
18)	(10,19)	1	1	2	0	123 213	11
19)	(16,15)	1	1	2	0	312 213	14
20)	(16,15)	1	1	2	0	231 321	12

SumFitness value of generation 1 = 245.0000
Max. Fitness value of generation 1 = 14.0000
Min. Fitness value of generation 1 = 11.0000
Avg. Fitness value of generation 1 = 12.2500
Cum. of NCross until gen. 1 = 9
Cum. of NMutation until gen. 1 = 0

Gambar 7.4 Populasi baru P(1) pada kasus 1

Proses yang mungkin terjadi pada suatu populasi, selain proses penyilangan adalah proses mutasi. Peluang terjadinya mutasi adalah sangat kecil apabila dibandingkan dengan peluang terjadinya penyilangan. Pada Kasus 1, peluang mutasi adalah sebesar 0,01, yang artinya adalah 1% kromosom dalam populasi baru mengalami mutasi. Mutasi terjadi hanya pada salah satu sub-kromosom dan salah satu gen.

Pada Kasus 1, mutasi terjadi pada generasi ke-7 atau populasi P(7) dan terjadi pada kromosom(10). Kromosom(10) adalah kromosom

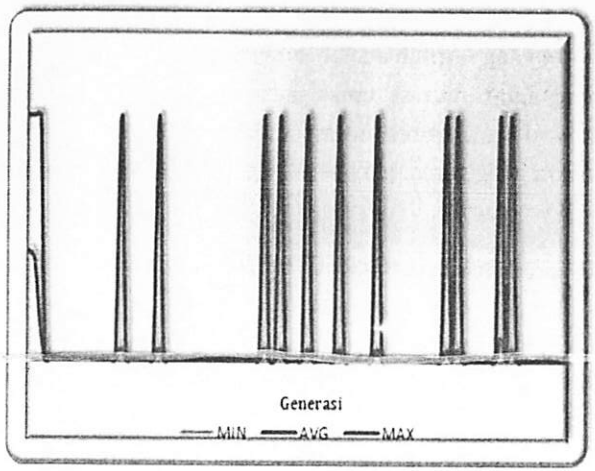
anak hasil penyilangan kromosom(10) dan kromosom(1) pada generasi ke-6 atau populasi P(6). Proses mutasi terjadi setelah proses penyilangan dan disebut dengan *mutation-embedded within crossover*. Berikut adalah kromosom-kromosom populasi P(6) sebagai kromosom induk populasi P(7) yang akan mengalami mutasi setelah mengalami proses penyilangan:

Generation 6

	Job		Makespan
1)	123	123	11
2)	123	123	11
3)	123	213	11
4)	123	123	11
5)	123	123	11
6)	123	123	11
7)	123	123	11
8)	123	123	11
9)	213	213	11
10)	123	123	11
11)	123	123	11
12)	213	123	11
13)	123	123	11
14)	213	123	11
15)	123	123	11
16)	123	123	11
17)	123	213	11
18)	123	123	11
19)	123	123	11
20)	213	123	11

Gambar 7.5 Kromosom-kromosom pada populasi P(6) sebelum mengalami penyilangan dan mutasi di populasi P(7)

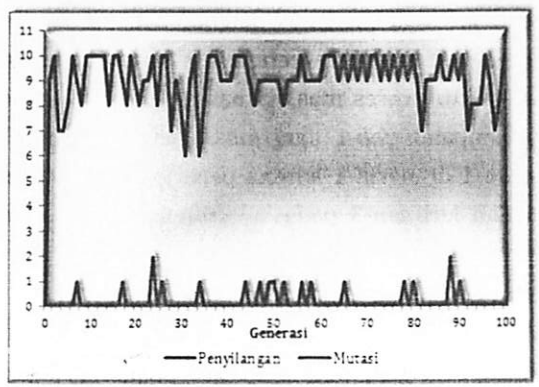
Berikut adalah populasi di mana pertama kali terjadi proses mutasi (Kasus1):



Gambar 7.8 Grafik nilai *makespan* maksimum, *makespan* rata-rata, dan *makespan* minimum untuk 100 generasi pada kasus 1 (*Pc=0,9; Pm=0,01; Bilangan Acak=0,6; PopSize=20, MaxGen=100*)

Nilai *makespan* maksimum tertinggi yang pernah diperoleh sepanjang 100 generasi adalah sebesar 14 dan nilai *makespan* minimum terendah yang pernah diperoleh sepanjang 100 generasi adalah sebesar 11. Nilai *makespan* mulai konvergen pada generasi ke-3, kemudian nilai 14 muncul kembali di generasi ke-17. Nilai tersebut muncul kembali dikarenakan oleh proses mutasi yang terjadi di generasi ke-17. Generasi selanjutnya kembali konvergen di nilai 11.

Semua nilai 14 yang kembali muncul setelah generasi ke-3 adalah hasil proses mutasi, meskipun tidak semua proses mutasi mengubah nilai *makespan* menjadi 14. Terdapat beberapa proses mutasi pada Kasus 1 yang tidak menyebabkan terjadinya perubahan pada nilai *makespan*.



Gambar 7.9 Grafik frekuensi penyilangan dan mutasi untuk 100 generasi pada Kasus 1 (*Pc=0,9; Pm=0,01; Bilangan Acak=0,6; PopSize=20, MaxGen=100*)

Sepanjang 100 generasi pada Kasus 1 telah terjadi proses penyilangan sebanyak 906 kali dan proses mutasi sebanyak 19 kali. Generasi ke-0 memang belum ada proses penyilangan dan mutasi. Kedua proses tersebut mulai dilakukan pada generasi ke-1.

Frekuensi tertinggi untuk penyilangan pada tiap-tiap generasi adalah sebanyak 10 kali dan frekuensi terendah sebanyak 6 kali. Mutasi dengan frekuensi tertinggi yaitu sebanyak 2 kali dan frekuensi terendah yaitu sebanyak 0 kali, artinya tidak dilakukan. Nilai *makespan* terbaik (paling optimum) yang diperoleh dari 100 generasi adalah 11, dengan struktur kromosomnya adalah [2 1 3] [2 1 3].

Pada tahap 1, yaitu penempatan *job-n* yang melewati mesin-*m* pada urutan proses ke-1. Apabila *job* tersebut merupakan urutan proses ke-1 pada mesin-*m* (sub-kromosom ke-*m*), penempatan dilakukan. Pengecekan dimulai dari gen yang pertama (paling kiri) dari masing-masing subkromosom. Penempatan didahulukan dari gen yang lebih awal (dari kiri ke kanan) pada subkromosom yang sama.

Job-1 dan *job*-3 di mesin-1; dan *job*-2 di mesin-2 merupakan urutan proses ke-1. Tempatkan ketiga *job* tersebut pada mesin yang sesuai dengan waktu proses masing-masing *job* pada mesin tersebut. Dahulukan penempatan *job*-1 dari *job*-3 yang keduanya merupakan urutan proses ke-1 di mesin-1 karena penerjemahan kromosom gen-1 berada di sebelah kiri gen-3 pada subkromosom ke-1. Pada tahap 2, yaitu penempatan *job-n* yang melewati mesin-*m* pada urutan proses ke-2. Penempatan pada tahap 2 hampir sama dengan tahap 1. Terdapat beberapa ketentuan dalam penempatan *job*, yaitu sebagai berikut.

Penempatan *job* dilakukan setelah *job* lain pada urutan proses sebelumnya di mesin yang sama selesai dilakukan, artinya satu mesin tidak dapat memproses beberapa *job* sekaligus, contohnya pada penempatan *job*-1 di mesin-2 dan *job*-2 di mesin-1.

Penempatan *job* yang sama pada mesin yang berbeda dilakukan setelah *job* tersebut pada urutan proses sebelumnya selesai dilakukan, artinya tiap-tiap *job* tidak dapat diproses pada satu waktu sekaligus di mesin yang berbeda, contohnya pada penempatan *job*-3.

7.3.2 Kasus 2: Penjadwalan *job shop* kasus 3 *job*-3 mesin

Pada Kasus 2, yaitu masalah penjadwalan 3 *job*-3 mesin merupakan studi literatur. Contoh kasus diperoleh dari buku *Genetic Algorithms and Engineering Design* (Gen dan Cheng 1997). Berikut adalah contoh kasusnya:

Tabel 2 Waktu proses, urutan mesin, dan urutan proses pada kasus 2 (3 *job*-3 mesin) (Gen dan Cheng 1997)

Waktu Proses				Urutan Mesin			
Job	Urutan Proses			Job	Urutan Proses		
	1	2	3		1	2	3
j_1	3	3	2	j_1	m_1	m_2	m_3
j_2	1	5	3	j_2	m_1	m_3	m_2
j_3	3	2	3	j_3	m_2	m_1	m_3

a. *Input*

Nilai-nilai utama yang diinput pada Kasus 2 adalah:

- Jumlah *job* = 3
- Jumlah mesin = 3
- Waktu proses dari tiap-tiap *job*
- Urutan proses dari tiap-tiap *job*

b. Parameter-parameter Algoritma Genetika

Nilai-nilai parameter algoritma genetika yang digunakan untuk Kasus 2 adalah sebagai berikut:

- Peluang penyilangan (*Pc*) = 0,9
- Peluang mutasi (*Pm*) = 0,01
- Bilangan acak (0–1) = 0,6
- Jumlah populasi (*PopSize*) = 20
- Jumlah generasi maksimum = 100

c. Inisialisasi Populasi Awal

Populasi awal P(0) dibangkitkan secara acak oleh sistem ***GA_JobShop*** dan hasil *running* dapat dilihat pada Gambar 7.10.

Pada populasi ke-0 nilai *makespan* tertinggi yaitu sebesar 14 dan terendah sebesar 10. Nilai *makespan* rata-rata pada populasi awal tersebut yang telah dibangkitkan oleh bilangan acak sebesar 0,5 adalah 11,15.

```
INITIAL REPORT
Simple Function Optimization with GAs
JOB SHOP SCHEDULING

GAs parameters
Pop. Size              = 20
Chrom. Length          = 9
Max. Generation        = 100
Crossover probability  = 0.9000
Mutation probability   = 0.0100

INITIAL GENERATION STASTISTICS
        Job          Makespan
 1) 312 132 231        14
 2) 213 132 123        12
 3) 213 213 231        11
 4) 213 321 231        11
 5) 231 123 213        12
 6) 123 132 321        14
 7) 321 321 321        11
 8) 123 213 123        14
 9) 321 321 231        11
10) 213 213 321        11
11) 132 213 123        14
12) 123 213 123        14
13) 132 213 213        14
14) 123 132 231        14
15) 132 321 312        14
16) 123 213 231        14
17) 312 213 213        14
18) 231 213 231        11
19) 231 321 321        11
20) 321 123 312        11

Sum of Fitness =   252.0000
Max. Fitness   =    14.0000
Min. Fitness   =    11.0000
Avg. Fitness   =    12.6000
```

Gambar 7.10 Populasi awal P(0) untuk kasus 2

d. Evaluasi dan Seleksi Kromosom

Kromosom-kromosom yang terpilih sebagai kromosom induk P(1) hasil seleksi dari populasi P(0) adalah sebagai berikut:

Kromosom(4), *makespan*-nya → $M_4 = 11$

Kromosom(7), *makespan*-nya → $M_7 = 11$

Kromosom(8), *makespan*-nya → $M_8 = 14$

Kromosom(9), *makespan*-nya → $M_9 = 11$

Kromosom(10), *makespan*-nya → $M_{10} = 11$

Kromosom(12), *makespan*-nya → $M_{12} = 14$

Kromosom(14), *makespan*-nya → $M_{14} = 14$

Kromosom(15), *makespan*-nya → $M_{15} = 14$

Kromosom(16), *makespan*-nya → $M_{16} = 14$

Kromosom(17), *makespan*-nya → $M_{17} = 14$

Kromosom(18), *makespan*-nya → $M_{18} = 11$

Kromosom(19), *makespan*-nya → $M_{19} = 11$

e. Penyilangan dan Mutasi

Pada kasus 2 peluang penyilangan adalah 0,9 artinya diharapkan 90% populasi yang terbentuk pada generasi berikutnya adalah hasil penyilangan generasi sebelumnya. Teknik penyilangan yang digunakan adalah metode *Partially Mapped Crossover* (PMX) yang telah dimodifikasi khusus untuk kromosom yang terdiri atas beberapa subkromosom. Berikut populasi baru P(1) hasil penyilangan kromosom-kromosom induk yang terseleksi:

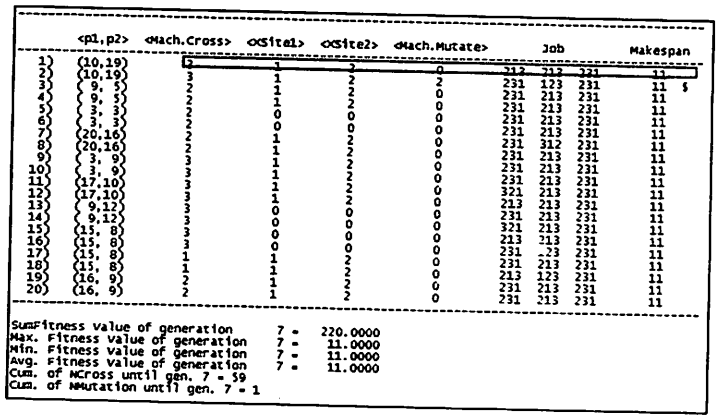
```
     <p1,p2>  <Mach.Cross>  <xSite1>  <xSite2>  <Mach.Mutate>      Job       Makespan
 1)  (16,17)        1          1         2          0        213 213 231      11
 2)  (16,17)        1          1         2          0        321 213 213      12
 3)  (16, 7)        2          1         2          0        123 123 231      14
 4)  (16, 7)        2          1         2          0        321 312 321      11
 5)  ( 9,18)        2          1         2          0        321 312 231      11
 6)  ( 9,18)        2          1         2          0        231 123 231      11
 7)  ( 9, 4)        3          1         2          0        321 321 231      11
 8)  ( 9, 4)        3          1         2          0        213 321 231      11
 9)  (10,19)        2          1         2          0        213 123 321      11
10)  (10,19)        2          1         2          0        231 312 321      11
11)  (10,15)        1          1         2          0        231 213 321      11
12)  (10,15)        1          1         2          0        312 321 312      14
13)  (14,18)        3          1         2          0        123 132 231      14
14)  (14,18)        3          1         2          0        231 213 231      11
15)  (18,14)        2          1         2          0        231 231 231      11
16)  (18,14)        2          1         2          0        123 312 231      14
17)  ( 9, 8)        3          1         2          0        321 321 321      11
18)  ( 9, 8)        3          1         2          0        123 213 132      14
19)  (12, 9)        2          1         2          0        123 123 123      14
20)  (12, 9)        2          1         2          0        321 312 231      11

SumFitness Value of generation     1 =   239.0000
Max. Fitness Value of generation   1 =    14.0000
Min. Fitness Value of generation   1 =    11.0000
Avg. Fitness Value of generation   1 =    11.9500
Cum. of NCross until gen. 1 = 10
Cum. of NMutation until gen. 1 = 0
```

Gambar 7.11 Populasi P(1) pada kasus 2

Proses mutasi pertama kali terjadi pada kromosom(2) yang terdapat di dalam generasi ke-7 atau populasi P(7). Kromosom tersebut mengalami *mutation-embedded within crossover* atau mutasi yang terjadi setelah proses penyilangan. Kromosom tersebut mengalami

mutasi pada subkromosom ke-2. Berikut adalah populasi P(7) yang di dalamnya terdapat kromosom yang pertama kali mengalami mutasi:



	<p1,p2>	<Mach.Cross>	<Site1>	<Site2>	<Mach.Mutate>	Job			Makespan
1)	(10,19)	2	1	2	0	213	312	231	11
2)	(10,19)	3	1	2	2	231	123	231	11
3)	(9, 5)	2	1	2	0	231	213	231	11
4)	(9, 5)	2	1	2	0	231	213	231	11
5)	(3, 3)	2	0	0	0	231	213	231	11
6)	(3, 3)	2	0	0	0	231	213	231	11
7)	(20,16)	2	1	2	0	231	312	231	11
8)	(20,16)	2	1	2	0	231	312	231	11
9)	(3, 9)	3	1	2	0	231	213	231	11
10)	(3, 9)	3	1	2	0	231	213	231	11
11)	(17,10)	3	1	2	0	231	213	231	11
12)	(17,10)	3	1	2	0	321	213	231	11
13)	(9,12)	3	0	0	0	213	213	231	11
14)	(9,12)	3	0	0	0	231	213	231	11
15)	(15, 8)	3	0	0	0	321	213	231	11
16)	(15, 8)	3	0	0	0	213	213	231	11
17)	(15, 8)	1	1	2	0	231	213	231	11
18)	(15, 8)	1	1	2	0	231	213	231	11
19)	(16, 9)	2	1	2	0	213	123	231	11
20)	(16, 9)	2	1	2	0	231	213	231	11

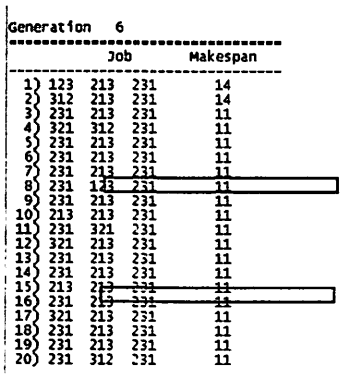
SumFitness value of generation 7 = 220.0000
Max. Fitness value of generation 7 = 11.0000
Min. Fitness value of generation 7 = 11.0000
Avg. Fitness value of generation 7 = 11.0000
Cum. of NCross until gen. 7 = 59
Cum. of NMutation until gen. 7 = 1

Gambar 7.12 Populasi P(7) terjadi mutasi pada kasus 2

Kromosom(1) dan kromosom(2) pada P(7) merupakan kromosom anak yang berasal dari sepasang kromosom pada populasi sebelumnya P(6), yaitu kromosom(10) dan kromosom(19). Keduanya telah mengalami proses penyilangan di P(7), yaitu pada subkromosom ke-3 dan titik penyilangan 1 dan 2. Kromosom induk dari populasi P(6) dapat dilihat pada Gambar 7.13.

Proses penyilangan terjadi antara kromosom(10) dan kromosom(19) pada subkromosom ke-3. Struktur kromosom(10) yaitu [2 1 3][2 1 3][2 3 1] dan kromosom(19) memiliki struktur [2 3 1][2 1 3][2 3 1]. Dapat dilihat bahwa subkromosom ke-3 pada kromosom(10) memiliki struktur yang sama dengan subkromosom ke-3 pada kromosom(19), akibatnya proses penyilangan menghasilkan kromosom anak yang identik dengan kromosom induk. Proses-proses algoritma genetika pada Kasus 2 mulai dari inisialisasi populasi awal, evaluasi kromosom, seleksi kromosom, proses penyilangan, serta proses mutasi

akan dihentikan apabila pencarian telah mencapai 100 generasi (*MaxGen* = 100).



Generation 6

	Job			Makespan
1)	123	213	231	14
2)	312	213	231	14
3)	231	213	231	11
4)	321	312	231	11
5)	231	213	231	11
6)	231	213	231	11
7)	231	213	231	11
8)	231	123	231	11
9)	231	213	231	11
10)	213	213	231	11
11)	231	321	231	11
12)	321	213	231	11
13)	231	213	231	11
14)	231	213	231	11
15)	213	213	231	11
16)	231	213	231	11
17)	321	213	231	11
18)	231	213	231	11
19)	231	213	231	11
20)	231	312	231	11

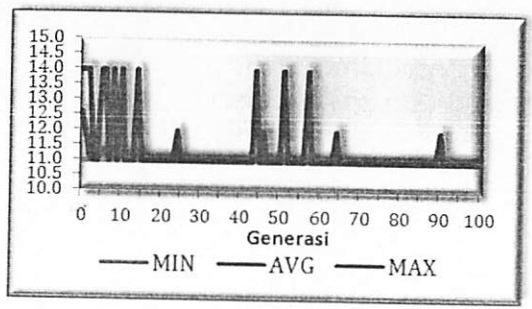
Gambar 7.13 Kromosom-kromosom induk pada populasi P(6) yang akan diseleksi untuk disilangkan dan dimutasi pada populasi P(7)

f. Hasil Running Sistem GA_JobShop

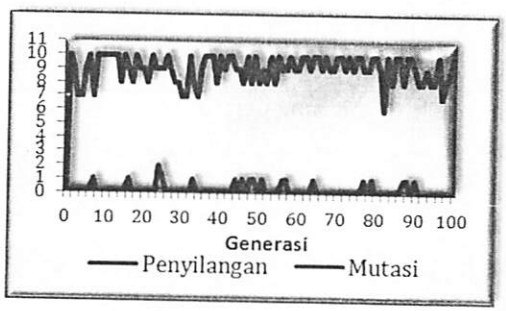
Hasil *running* sistem *GA_JobShop* untuk Kasus 2 diilustrasikan dalam dua buah grafik. Grafik yang pertama yaitu terdapat pada Gambar 7.14, menunjukkan nilai *makespan* maksimum (*Max*), *makespan* rata-rata (*Avg*), dan *makespan* minimum (*Min*) dari generasi awal (generasi ke-0) hingga generasi maksimum (generasi ke-100). Grafik berikutnya terdapat pada Gambar 7.15, menunjukkan frekuensi terjadinya penyilangan dan mutasi sepanjang 100 generasi.

Gambar 7.14 memperlihatkan bahwa nilai *makespan* maksimum tertinggi yang pernah diperoleh sepanjang 100 generasi adalah sebesar 14 dan nilai *makespan* minimum terendah yang pernah diperoleh sepanjang 100 generasi adalah sebesar 11. Nilai *makespan* mulai konvergen pada generasi ke-3, tetapi terjadi beberapa kali peningkatan nilai *makespan* rata-rata yang dikarenakan nilai-nilai yang lebih tinggi

dari nilai minimum bermunculan kembali. Hal tersebut diakibatkan oleh proses penyilangan dan mutasi yang terjadi. Pada generasi ke-5, nilai *makespan* 14 kembali muncul akibat proses penyilangan.



Gambar 7.14 Grafik nilai *makespan* maksimum, *makespan* rata-rata, dan *makespan* minimum untuk 100 generasi pada kasus 2 (*Pc=0,9; Pm=0,0; Bilangan Acak=0,6; PopSize=20, MaxGen=100*)



Gambar 7.15 Grafik frekuensi penyilangan dan mutasi untuk 100 generasi pada kasus 2 (*Pc=0,9; Pm=0,01; Bilangan Acak=0,6; PopSize=20, MaxGen=100*)

Terjadi beberapa ketidakstabilan nilai sepanjang 100 generasi meskipun nilai minimum sempat konvergen pada generasi ke-3. Nilai minimum mulai stabil pada generasi ke-16 dan tidak stabil lagi pada

generasi ke-24 yang diakibatkan oleh proses mutasi. Nilai rata-rata kembali stabil pada generasi ke-25 hingga generasi ke-43, generasi ke-65 hingga generasi ke-89, dan generasi ke-91 hingga generasi ke-100. Ketidakstabilan yang terjadi setelah nilai konvergen yaitu akibat proses mutasi yang memunculkan kembali nilai-nilai maksimum yang sempat punah pada generasi sebelumnya.

Sepanjang 100 generasi pada Kasus 2 telah terjadi proses penyilangan sebanyak 907 kali dan proses mutasi sebanyak 19 kali. Kedua proses tersebut mulai dilakukan pada generasi ke-1.

Frekuensi tertinggi untuk penyilangan pada tiap-tiap generasi adalah sebanyak 10 kali dan frekuensi terendah sebanyak 6 kali. Mutasi dengan frekuensi tertinggi yaitu sebanyak 2 kali hanya terjadi sekali pada generasi ke-24 dan frekuensi terendah yaitu sebanyak 0 kali (tidak dilakukan). Hal ini dikarenakan mutasi tidak harus selalu terjadi pada tiap generasi.

Nilai *makespan* terbaik (paling optimum) yang diperoleh dari 100 generasi adalah 11, dengan hasil akhir struktur kromosomnya adalah [2 3 1] [1 2 3] [2 3 1] atau [2 3 1] [3 2 1] [2 3 1] atau [2 3 1] [1 3 2] [2 3 1].

g. Efisiensi Algoritma Genetika

Pada Kasus 2, yaitu 3 *job* dan 3 mesin memiliki *total search space* atau jumlah calon solusi dalam ruang pencarian (*Nt*) sebagai berikut:

$Nt = (3!)^3 = (3\times2\times1)^3 =$ **216 calon solusi**

Solusi optimum penjadwalan *job shop* untuk Kasus 2 mulai konvergen pada generasi ke-3, artinya nilai optimum yang ditemukan telah konstan pada generasi ke-3 dan tidak ada lagi nilai yang lebih kecil dari nilai tersebut. Jumlah calon solusi yang telah dievaluasi oleh algoritma genetika hingga konvergen (*Ns*) adalah sebagai berikut:

Ns = (3 generasi) x (20 calon solusi/generasi) = **60 calon solusi**

Persentase pencarian calon solusi yang dilakukan algoritma genetika dalam ruang pencarian (*Ps*) adalah:

$$Ps = (Ns/Nt) \times 100\ \%$$

$$= (60/216) \times 100\ \% = 27{,}78\ \%$$

Persentase tersebut menunjukkan bahwa algoritma genetika hanya mengeksplorasi sebesar 27,78% dari ruang pencarian untuk mencari solusi penjadwalan *job shop* yang optimum. Hal ini menunjukkan bahwa algoritma genetika sudah bekerja secara efisien pada kasus penjadwalan 3 *job*-3 mesin.

7.3.3 Kasus 3: Penjadwalan *job shop* kasus 5 *job*-12 mesin

Pada Kasus 3, data diperoleh dari kombinasi antara data sekunder yang diambil dari Riswan (1993) dan Aryawan (2003), dengan pengamatan langsung yang dilakukan di CV Mitra Niaga Indonesia, Bogor. Data yang diambil dari Riswan (1993) dan Aryawan (2003) disajikan pada Tabel 7.3, Tabel 7.4, dan Tabel 7.5. Tujuan penggunaan kombinasi data sekunder dan data primer ini adalah untuk memperlihatkan aplikasi metode algoritma genetika ini di kasus nyata industri peralatan pengolahan hasil pertanian. Perusahaan yang dikunjungi untuk pengambilan data primer adalah industri yang memproduksi alat dan mesin pengolahan hasil pertanian. Produk-produk yang dihasilkan di antaranya adalah *destilator, vacuum drying, mixer,* pemarut kelapa, pencacah kompos, dan lain-lain. Pada proses produksinya, industri tersebut memiliki penjadwalan bertipe *job shop.*

Sebuah alat atau mesin pengolahan hasil pertanian merupakan serangkaian yang terdiri atas beberapa komponen. Masing-masing komponen tersebut memiliki proses produksi yang berbeda-beda. Proses pembuatan masing-masing komponen tersebut disebut juga dengan *job.* Masing-masing *job* memiliki lintasan atau jalur yang harus

ditempuh dalam satu siklus produksinya, tiap jalur adalah identik untuk satu *job* tertentu dan berisi urutan mesin-mesin yang harus dilewati *job* tersebut. Berikut adalah daftar urutan proses, urutan mesin, dan waktu proses untuk contoh kasus lima *job* yang harus diselesaikan.

Tabel 7.3 Waktu proses (satuan waktu menit), urutan mesin, dan urutan proses pada kasus 3 (5 *job*-12 mesin)

Waktu Proses								Urutan Mesin							
Job	Urutan Proses							*Job*	Urutan Proses						
	1	2	3	4	5	6	7		1	2	3	4	5	6	7
*j*1	340	146	555	188	355			*j*1	*m*1	*m*3	*m*5	*m*7	*m*10		
*j*2	506	171	220	881	248	415		*j*2	*m*2	*m*3	*m*7	*m*8	*m*11	*m*10	
*j*3	227	460	289	126	762	460		*j*3	*m*1	*m*6	*m*2	*m*7	*m*9	*m*6	
*j*4	574	297	361	157	629	177	296	*j*4	*m*6	*m*4	*m*2	*m*7	*m*8	*m*11	*m*10
*j*5	454	918	195	741	1007			*j*5	*m*1	*m*6	*m*5	*m*3	*m*8		

Dari data pada Tabel 7.3 dapat dilihat bahwa jumlah proses maksimum sebanyak tujuh proses. *Job* 1 dan *job* 5 terdiri atas lima proses, *job* 2 dan *job* 3 terdiri atas enam proses, dan *job* 4 terdiri atas tujuh proses. Masing-masing proses memiliki waktu proses yang berbeda-beda.

Mesin yang digunakan untuk proses produksi *job-job* tersebut adalah sebanyak 11 jenis mesin. Pada *job* 3 melewati mesin 6 sebanyak dua kali, yaitu pada proses kedua dan keenam. Sistem ***GA_JobShop*** yang dibuat hanya bisa memproses masing-masing *job* melewati mesin yang sama tidak lebih dari satu kali, sehingga mesin 6 pada proses yang kedua kalinya (proses keenam) dapat dianggap sebagai mesin 12. Berikut adalah nama-nama *job* yang akan diproduksi:

Tabel 7.4 Daftar nama-nama *job* pada kasus 3

No.Job	Nama Job
j1	*Inlet Oil Cooler*
j2	*Foam Chassis*
j3	*Power S Cord 1.25*
j4	*Eva Cover*
j5	*Condensor Filter*

Berikut adalah daftar nama-nama mesin yang digunakan pada proses produksi:

Tabel 7.5 Daftar nama-nama mesin pada kasus 3

No.Mesin	Nama Mesin	No.Mesin	Nama Mesin
m1	Mesin Potong Lurus	m7	Mesin *Punch*
m2	Mesin Potong Alur	m8	Mesin *Las Spot*
m3	Mesin *Press* 20 ton	m9	Mesin *Las Arc*
m4	Mesin *Press* 40 ton	m10	Mesin Gerinda
m5	Mesin *Press* 60 ton	m11	Mesin Bor
m6	Mesin *Press* 80 ton	m12	Mesin *Press* 80 ton

Job 1, yaitu produksi *inlet oil cooler,* pada prosesnya harus melewati lima mesin (m_1- m_3- m_5- m_7- m_{10}). Pada tahapan proses yang pertama, *job* 1 diproses di mesin 1 selama 340 menit. Kemudian, pada tahapan proses yang kedua, *job* 1 diproses di mesin 3 selama 146 menit. Tahapan selanjutnya, yaitu proses yang ketiga, *job* 1 diproses di mesin 5 selama 555 menit. Pada tahapan proses yang keempat, *job* 1 diproses di mesin 7 selama 188 menit. Terakhir, *job* 1 diproses di mesin 10 selama 355 menit.

Job 2, yaitu produksi *foam chassis,* pada prosesnya harus melewati enam mesin (m_2- m_3- m_7- m_8- m_{11}- m_{10}). *Job* 3, yaitu produksi *power s cord 1.25,* harus melewati enam mesin pula namun memiliki urutan proses yang berbeda dari *job* 2 (m_1- m_6- m_2- m_7- m_9- m_{12}). Proses

produksi *job* 4, yaitu *eva cover* harus melewati tujuh proses (m_6- m_4- m_2- m_7- m_8- m_{11}- m_{10}). *Job* 5, yaitu *condenser filter,* pada prosesnya harus melewati lima mesin (m_1- m_6- m_5- m_3- m_8). Masing-masing *job* memiliki tahapan proses yang berbeda dan waktu proses yang berbeda pula.

a. *Input*

Nilai-nilai utama yang di-*input* pada Kasus 3 adalah:

- Jumlah job = 5
- Jumlah mesin = 12
- Waktu proses dari tiap-tiap *job*
- Urutan proses dari tiap-tiap *job*

Waktu proses dan urutan proses di-*input* sesuai dengan data pada Tabel 7.3. Jumlah proses maksimum yang terdapat pada sistem ***GA_JobShop*** untuk masing-masing *job* adalah sebanyak jumlah mesin, artinya apabila kasus yang akan dipecahkan terdapat 12 mesin, urutan proses maksimum yang tersedia adalah sebanyak 12 proses, meskipun proses maksimum yang terjadi pada Kasus 3 adalah sebanyak tujuh proses.

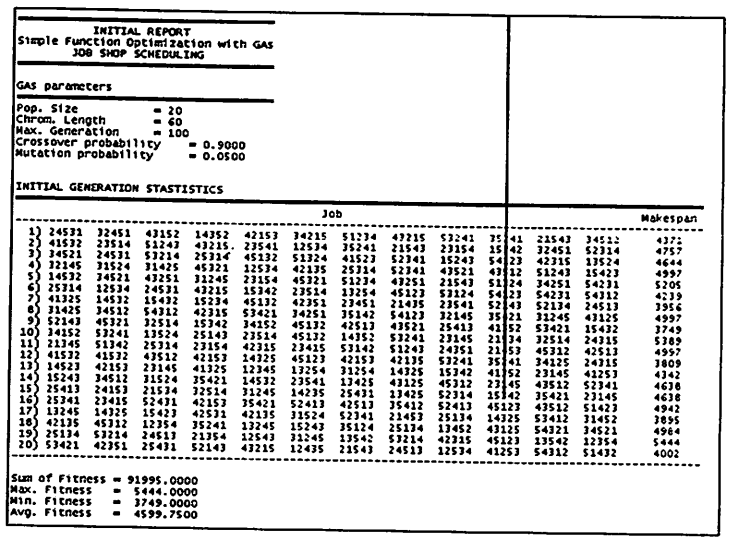
b. Parameter-parameter Algoritma Genetika

Nilai-nilai parameter algoritma genetika yang digunakan untuk Kasus 3 adalah sebagai berikut:

- Peluang penyilangan (*Pc*) = 0,9
- Peluang mutasi (*Pm*) = 0,05
- Bilangan acak (0–1) = 0,5
- Jumlah populasi (*PopSize*) = 20
- Jumlah generasi maksimum = 100

c. Inisialisasi Populasi Awal

Populasi awal P(0) dibangkitkan secara acak oleh sistem ***GA_JobShop*** dan hasil *running* dapat dilihat pada Gambar 7.16.



```
                INITIAL REPORT
Simple Function Optimization with GAS
          JOB SHOP SCHEDULING

GAS parameters

Pop. Size          = 20
Chrom. Length      = 60
Max. Generation    = 100
Crossover probability    = 0.9000
Mutation probability     = 0.0500

INITIAL GENERATION STASTISTICS

                                   Job                                  Makespan
 1) 24531 32451 43152 14352 42153 34215 51234 43215 53241 35141 21543 34512   4371
 2) 41532 23514 51243 43215 23541 12534 35241 21543 23154 15142 32451 52314   4757
 3) 34521 24531 53214 25314 45132 51324 41523 52341 15243 54123 42315 13524   4644
 4) 32145 31524 31425 45321 12534 42135 25314 52341 43521 43152 51243 15423   4997
 5) 14532 34521 43251 31245 23154 45321 51234 43251 21543 51324 34251 54231   5205
 6) 25314 12534 24531 43215 15342 23514 13254 45123 53124 54123 54231 54312   4239
 7) 41325 14532 15423 15234 45132 42351 23451 21435 23541 52143 52134 24513   3956
 8) 31425 34512 54312 42315 53421 34251 35142 54123 32145 35421 31245 43125   4997
 9) 52143 45321 32514 15342 34152 45132 42513 43521 25413 41352 53421 15422   3749
10) 34152 53241 13524 25143 23514 45132 14352 53241 23145 21534 32514 24315   5389
11) 21345 51342 25314 23154 42315 23415 53142 51243 24351 21453 45312 42513   4957
12) 41532 41532 43512 42153 14325 45123 42153 42135 53241 35141 34125 24315   3809
13) 14523 42153 23145 41325 12345 13254 31254 14325 15342 41352 23145 41253   4342
14) 15243 34512 31524 35421 14532 23541 13425 43125 45312 23145 43512 52341   4638
15) 25413 24153 21534 32514 31245 14235 25431 13425 52314 15142 35421 23145   4638
16) 25341 23415 52431 42153 35421 52413 42513 35412 52413 45123 43512 51423   4942
17) 13245 14325 15423 42531 42135 31524 52341 23453 25134 14325 53412 31452   3895
18) 42135 45312 12354 35241 13245 15243 35124 25134 13452 43125 54321 34521   4984
19) 25134 53214 24513 21354 12543 31245 13542 53214 42315 45123 13542 12354   5444
20) 53421 42351 25431 52143 43215 12435 21543 24513 12534 41253 54312 51432   4002

Sum of Fitness = 91995.0000
Max. Fitness   =  5444.0000
Min. Fitness   =  3749.0000
Avg. Fitness   =  4599.7500
```

Gambar 7.16 Populasi awal P(0) untuk kasus 3

d. Evaluasi dan Seleksi Kromosom

Kromosom-kromosom yang terpilih sebagai kromosom induk P(1) hasil seleksi dari populasi P(0) adalah sebagai berikut:

Kromosom(1), *makespan*-nya → M_1 = 4371

Kromosom(2), *makespan*-nya → M_2 = 4757

Kromosom(6), *makespan*-nya → M_6 = 4239

Kromosom(7), *makespan*-nya → M_7 = 3956

Kromosom(11), *makespan*-nya → M_{11} = 4997

Kromosom(12), *makespan*-nya → M_{12} = 3809

Kromosom(13), *makespan*-nya → M_{13} = 4342

Kromosom(15), *makespan*-nya → M_{15} = 4638

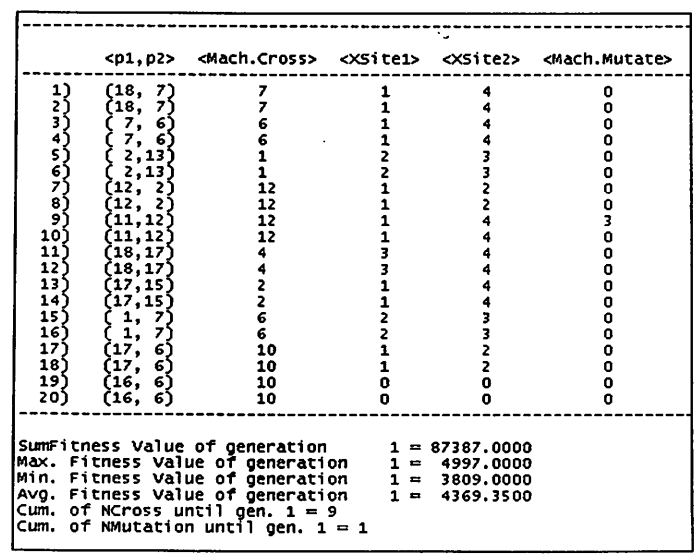
Kromosom(16), *makespan*-nya → M_{16} = 4942

Kromosom(17), *makespan*-nya → M_{17} = 3895

Kromosom(18), *makespan*-nya → M_{18} = 4948

e. Penyilangan dan Mutasi

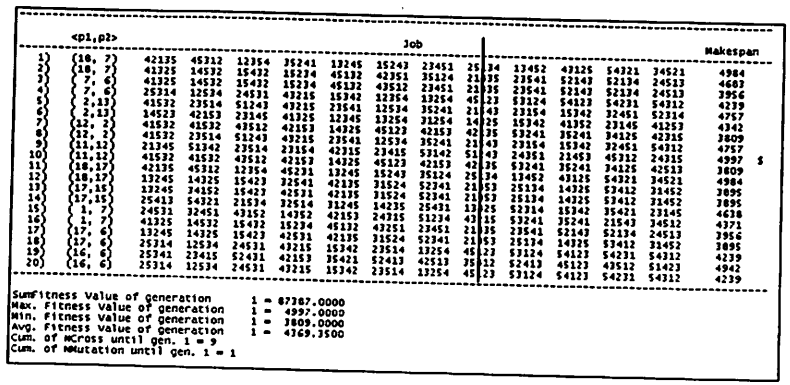
Pada kasus 3 peluang penyilangan adalah 0,9 artinya diharapkan 90% populasi yang terbentuk pada generasi berikutnya adalah hasil penyilangan generasi sebelumnya. Teknik penyilangan yang digunakan adalah metode *Partially Mapped Crossover* (PMX) yang telah dimodifikasi khusus untuk kromosom yang terdiri atas beberapa subkromosom. Berikut kromosom-kromosom yang terpilih P(0) dan akan mengalami proses penyilangan.



```
      <p1,p2>   <Mach.Cross>  <XSite1>  <XSite2>  <Mach.Mutate>
 1)  (18, 7)         7          1         4          0
 2)  (18, 7)         7          1         4          0
 3)  ( 7, 6)         6          1         4          0
 4)  ( 7, 6)         6          1         4          0
 5)  ( 2,13)         1          2         3          0
 6)  ( 2,13)         1          2         3          0
 7)  (12, 2)        12          1         2          0
 8)  (12, 2)        12          1         2          0
 9)  (11,12)        12          1         4          3
10)  (11,12)        12          1         4          0
11)  (18,17)         4          3         4          0
12)  (18,17)         4          3         4          0
13)  (17,15)         2          1         4          0
14)  (17,15)         2          1         4          0
15)  ( 1, 7)         6          2         3          0
16)  ( 1, 7)         6          2         3          0
17)  (17, 6)        10          1         2          0
18)  (17, 6)        10          1         2          0
19)  (16, 6)        10          0         0          0
20)  (16, 6)        10          0         0          0

SumFitness Value of generation      1 = 87387.0000
Max. Fitness Value of generation    1 =  4997.0000
Min. Fitness Value of generation    1 =  3809.0000
Avg. Fitness Value of generation    1 =  4369.3500
Cum. of NCross until gen. 1 = 9
Cum. of NMutation until gen. 1 = 1
```

Gambar 7.17 Populasi P(1) pada kasus 3 (bagian 1)

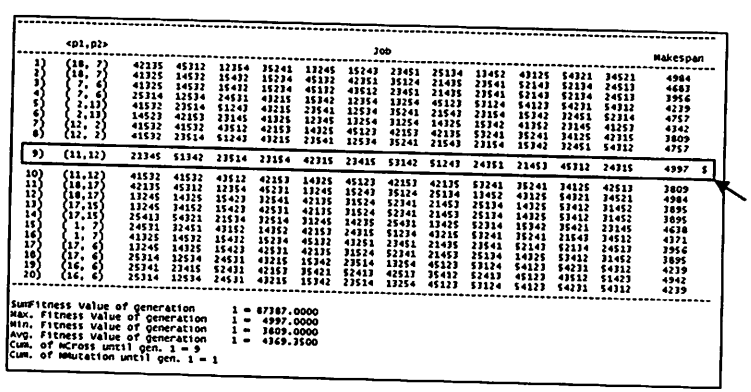


```
      <p1,p2>                                                    Job                                                    Makespan
  1)  (10, 7)  42135  45312  12354  35241  13245  15243  23451  25134  13452  43125  54321  34521   4984
  2)  (10, 7)  41325  14532  15432  15234  45132  42351  35124  21435  23541  52143  52334  24513   4683
  3)  ( 7, 6)  41325  14532  15432  15234  45132  43512  23451  21335  23541  52143  52334  24513   3956
  4)  ( 7, 6)  25314  12534  24531  43215  15342  12354  13254  45123  53124  54123  54231  54312   4239
  5)  ( 2,13)  41532  23514  51243  43215  23541  12534  35241  21543  23154  15342  32451  52314   4757
  6)  ( 2,13)  14523  42153  23145  43215  12345  13254  31254  14325  15342  41352  23145  41253   4342
  7)  (12, 2)  41532  41532  43512  42153  14325  45123  42153  42135  53241  35241  34125  42315   3809
  8)  (12, 2)  41532  23514  51243  43215  23541  12534  35241  21543  23154  15342  32451  54312   4757
  9)  (11,12)  21345  51342  23514  23154  42315  23415  53142  51243  24351  21453  45312  24315   4997  $
 10)  (11,12)  41532  41532  43512  42153  14325  45123  42153  42135  53241  35241  34125  42315   3809
 11)  (18,17)  42135  45312  12354  45231  13245  15243  35124  25134  13452  43125  54321  34521   4984
 12)  (18,17)  13245  14325  15423  32541  42135  31524  52341  21453  25134  14325  53412  31452   3895
 13)  (17,15)  13245  34152  15423  42531  42135  31524  52341  21453  25134  14325  53412  31452   3895
 14)  (17,15)  25413  54321  21534  32514  31245  14235  25431  13425  52314  15342  35421  23145   4638
 15)  ( 1, 7)  24531  32451  43152  14352  42153  24315  51234  43125  53241  35241  21543  34512   4371
 16)  ( 1, 7)  41325  14532  15432  15234  45132  42351  23451  21435  23541  52143  52334  24513   3956
 17)  (17, 6)  13245  14325  15423  42531  42135  31524  52341  21453  25134  14325  53412  31452   3895
 18)  (17, 6)  25314  12534  24531  43215  15342  23514  13254  45123  53124  54123  54231  54312   4239
 19)  (16, 6)  25341  23415  52431  42153  35421  52413  42513  35432  52413  45123  43512  51423   4942
 20)  (16, 6)  25314  12534  24531  43215  15342  23514  13254  45123  53124  54123  54231  54312   4239

SumFitness Value of generation      1 = 87387.0000
Max. Fitness Value of generation    1 =  4997.0000
Min. Fitness Value of generation    1 =  3809.0000
Avg. Fitness Value of generation    1 =  4369.3500
Cum. of nCross until gen. 1 = 9
Cum. of nMutation until gen. 1 = 1
```

Gambar 7.18 Populasi P(1) pada kasus 3 (bagian 2)

Proses mutasi pertama kali terjadi pada generasi ke-1 atau populasi P(1). Mutasi terjadi sebanyak satu kali pada populasi tersebut, yaitu pada kromosom(9). Kromosom-kromosom tersebut mengalami *mutation-embedded within crossover* atau mutasi yang terjadi setelah proses penyilangan. Kromosom(9) mengalami mutasi pada subkromosom ke-3. Berikut adalah populasi P(1) yang di dalamnya terdapat kromosom yang mengalami mutasi.



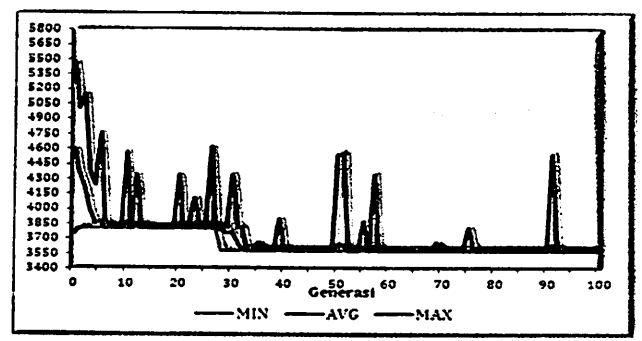
```
      <p1,p2>                                                    Job                                                    Makespan
  1)  (10, 7)  42135  45312  12354  35241  13245  15243  23451  25134  13452  43125  54321  34521   4984
  2)  (10, 7)  41325  14532  15432  15234  45132  42351  35124  21435  23541  52143  52334  24513   4683
  3)  ( 7, 6)  41325  14532  15432  15234  45132  43512  23451  21335  23541  52143  52334  24513   3956
  4)  ( 7, 6)  25314  12534  24531  43215  15342  12354  13254  45123  53124  54123  54231  54312   4239
  5)  ( 2,13)  41532  23514  51243  43215  23541  12534  35241  21543  23154  15342  32451  52314   4757
  6)  ( 2,13)  14523  42153  23145  43215  12345  13254  31254  14325  15342  41352  23145  41253   4342
  7)  (12, 2)  41532  41532  43512  42153  14325  45123  42153  42135  53241  35241  34125  42315   3809
  8)  (12, 2)  41532  23514  51243  43215  23541  12534  35241  21543  23154  15342  32451  54312   4757
  9)  (11,12)  23345  51342  23514  23154  42315  23415  53142  51243  24351  21453  45312  24315   4997  $
 10)  (11,12)  41532  41532  43512  42153  14325  45123  42153  42135  53241  35241  34125  42315   3809
 11)  (18,17)  42135  45312  12354  45231  13245  15243  35124  25134  13452  43125  54321  34521   4984
 12)  (18,17)  13245  14325  15423  32541  42135  31524  52341  21453  25134  14325  53412  31452   3895
 13)  (17,15)  13245  34152  15423  42531  42135  31524  52341  21453  25134  14325  53412  31452   3895
 14)  (17,15)  25413  54321  21534  32514  31245  14235  25431  13425  52314  15342  35421  23145   4638
 15)  ( 1, 7)  24531  32451  43152  14352  42153  24315  51234  43125  53241  35241  21543  34512   4371
 16)  ( 1, 7)  41325  14532  15432  15234  45132  42351  23451  21435  23541  52143  52334  24513   3956
 17)  (17, 6)  13245  14325  15423  42531  42135  31524  52341  21453  25134  14325  53412  31452   3895
 18)  (17, 6)  25314  12534  24531  43215  15342  23514  13254  45123  53124  54123  54231  54312   4239
 19)  (16, 6)  25341  23415  52431  42153  35421  52413  42513  35432  52413  45123  43512  51423   4942
 20)  (16, 6)  25314  12534  24531  43215  15342  23514  13254  45123  53124  54123  54231  54312   4239

SumFitness Value of generation      1 = 87387.0000
Max. Fitness Value of generation    1 =  4997.0000
Min. Fitness Value of generation    1 =  3809.0000
Avg. Fitness Value of generation    1 =  4369.3500
Cum. of nCross until gen. 1 = 9
Cum. of nMutation until gen. 1 = 1
```

Gambar 7.19 Populasi P(1) terjadi mutasi pada kasus 3

f. Hasil *Running* Sistem GA_JobShop

Hasil *running* sistem *GA_JobShop* untuk Kasus 3 diilustrasikan dalam dua buah grafik. Grafik yang pertama yaitu terdapat pada Gambar 7.20, menunjukkan nilai *makespan* maksimum (*Max*), *makespan* rata-rata (*Avg*), dan *makespan* minimum (*Min*) dari generasi awal (generasi ke-0) hingga generasi maksimum (generasi ke-100). Grafik berikutnya terdapat pada Gambar 7.21, menunjukkan frekuensi terjadinya penyilangan dan mutasi sepanjang 100 generasi.

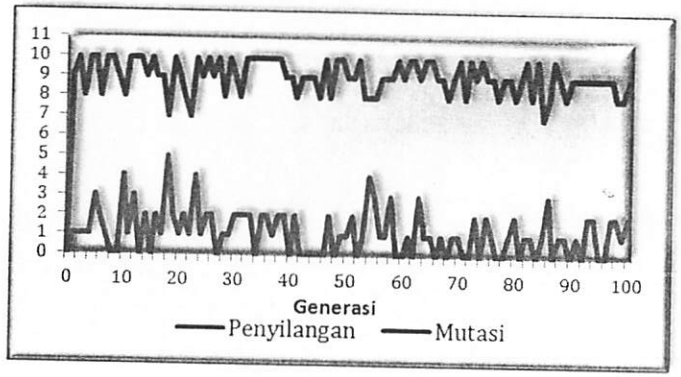


Gambar 7.20 Grafik nilai *makespan* maksimum, *makespan* rata-rata, dan *makespan* minimum (satuan waktu dalam menit) untuk 100 generasi pada kasus 3 (*Pc=0,9; Pm=0,05; Bilangan Acak=0,5; PopSize=20, MaxGen=100*)

Nilai *makespan* maksimum tertinggi terjadi pada generasi awal yaitu pada saat inisialisasi populasi awal sebesar 5.444 menit, setelah itu terjadi penurunan nilai *makespan* rata-rata yang berkelanjutan hingga konvergen di nilai minimum sebesar 3.589 menit pada generasi ke-33. Nilai minimum tersebut muncul pertama kali pada generasi ke-28. Pencapaian nilai optimum tersebut dikarenakan oleh proses mutasi yang terjadi sehingga menghasilkan kromosom baru yang lebih optimum nilainya.

Berdasarkan G menunjukkan bahwa setelah terjadi konvergensi nilai *makespan* yang minimum (berawal di generasi ke-40), beberapa generasi berikutnya mengalami peningkatan nilai *makespan* akibat mutan-mutan yang bermunculan. Peningkatan nilai *makespan* tersebut terjadi pada generasi ke-50, generasi ke-51, generasi ke-55, generasi ke-57, generasi ke-69, generasi ke-75, dan generasi ke-91.

Sepanjang 100 generasi pada Kasus 3 telah terjadi proses penyilangan sebanyak 909 kali dan proses mutasi sebanyak 120 kali. Proses keduanya dapat dilihat pada Gambar 7.21.



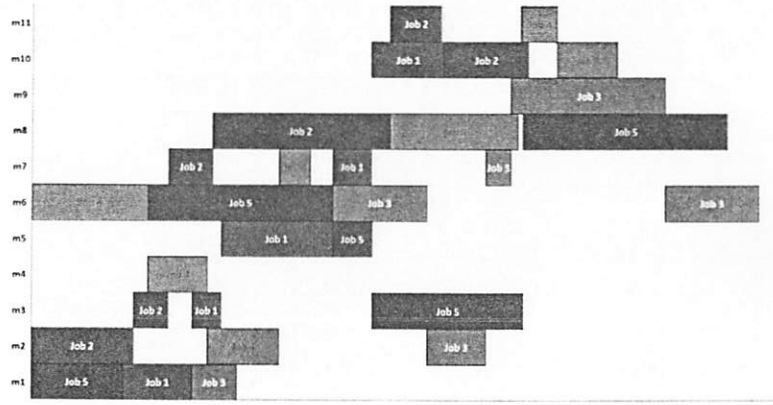
Gambar 7.21 Grafik frekuensi penyilangan dan mutasi untuk 100 generasi pada kasus 3 (*Pc=0,9; Pm=0,05; Bilangan Acak=0,5; PopSize=20, MaxGen=100*)

Frekuensi tertinggi untuk penyilangan pada tiap-tiap generasi adalah sebanyak 10 kali dan frekuensi terendah sebanyak 7 kali. Mutasi dengan frekuensi tertinggi yaitu sebanyak 5 kali dan frekuensi terendah yaitu sebanyak 0 kali (tidak dilakukan).

Jumlah kromosom yang mengalami proses mutasi pada Kasus 3 ini dipengaruhi oleh peluang mutasi (*Pm*) yang digunakan yaitu sebesar 0,05, artinya 5% dari tiap-tiap populasi akan mengalami proses mutasi. Pada proses pencarian, mutasi sangat berguna dalam pencarian nilai

yang lebih optimum. Namun, tidak selamanya demikian karena mutasi juga dapat menurunkan nilai *makespan* rata-rata pada sebuah populasi.

Nilai *makespan* terbaik (paling optimum) yang diperoleh dari 100 generasi adalah 3.589 menit, dengan struktur kromosomnya adalah [25134] [41532] [43521] [42153] [14352] [45123] [42513] [42135] [53142] [35241] [54132] [41523]. Berikut adalah gambar penjadwalan yang optimum hasil penerjemahan kromosom tersebut.



Gambar 7.22 Penjadwalan yang optimum pada kasus 3 (hasil penerjemahan kromosom terbaik)

g. Efisiensi Algoritma Genetika

Pada Kasus 3, yaitu 5 *job* dan 12 mesin memiliki *total search space* atau jumlah calon solusi dalam ruang pencarian (*Nt*) sebanyak:

$Nt = (5!)^{12} = 8{,}92 \times 10^{24}$ calon solusi

Solusi optimum penjadwalan *job shop* untuk Kasus 3 telah konvergen pada generasi ke-33, tidak ada lagi nilai yang lebih kecil dari nilai tersebut. Jumlah calon solusi yang telah dievaluasi oleh algoritma genetika hingga konvergen (*Ns*) adalah sebagai berikut:

Ns = (33 generasi) x (20 calon solusi/generasi) = **660 calon solusi**

Persentase pencarian calon solusi yang dilakukan algoritma genetika dalam ruang pencarian (Ps) adalah:

$$Ps = (Ns/Nt) \times 100\ \%$$
$$= (660/8{,}92 \times 10^{24}) \times 100\ \%$$

Persentase tersebut akan menghasilkan nilai yang teramat kecil. Semakin kecil persentase yang dihasilkan, pencarian nilai optimum akan semakin efisien. Hal tersebut membuktikan bahwa semakin besar nilai ruang pencarian (*search space*), efisiensi algoritma genetika akan semakin meningkat. Pencarian solusi optimum dengan algoritma genetika pada Kasus 3 terbukti sangat efisien.

7.4 Kesimpulan

Penjadwalan tipe *job shop* berskala besar memiliki tingkat kerumitan yang tinggi karena memerhatikan urutan proses dan kombinasi penjadwalan yang sangat banyak. Algoritma genetika, yaitu suatu pencarian acak yang meniru proses alam (evolusi biologi) mampu untuk memecahkan masalah penjadwalan tipe *job shop* berskala besar. Permasalahan dalam penjadwalan tipe *job shop* adalah mencari kombinasi urutan *job* yang tepat sehingga didapatkan nilai *makespan* yang paling minimum. Representasi kromosom menggunakan *preference list based representation*, proses penyilangan menggunakan *Partially Mapped Crossover* (PMX) yang telah dimodifikasi, proses mutasi menggunakan *reciprocal exchange mutation* yang telah dimodifikasi, dan seleksi menggunakan *tournament selection*.

Hasil menunjukkan bahwa sistem ***GA_JobShop*** sangat efisien dalam penyelesaian masalah penjadwalan tipe *job shop* berskala kecil maupun besar. Pada Kasus 2, solusi optimum dapat tercapai pada generasi ke-3 dengan nilai *makespan* 11. Algoritma genetika hanya

mengeksplorasi sebesar 27,78 % ruang pencarian (*search space*) untuk mendapatkan nilai yang minimum pada Kasus 2. Semakin besarnya ruang pencarian (*search space*), penggunaan algoritma genetika akan semakin efisien, terlihat pada Kasus 3. Kasus 3 (5 *job*-12 mesin) memiliki nilai *makespan* sebesar 3.589 dalam satuan waktu menit dan struktur kromosom terbaiknya adalah [25134] [41532] [43521] [42153] [14352] [45123] [42513] [42135] [53142] [35241] [54132] [41523]. Hasil kromosom tersebut kemudian diterjemahkan ke dalam gambar dengan metode *preference list based representation*. Terbukti pada Kasus 3 (skala besar) bahwa algoritma genetika mampu menemukan solusi optimum dengan sangat efisien.

Bab 8
Aplikasi Algoritma Genetika dalam Sistem Penunjang Keputusan Cerdas Bidang Agroindustri

Aplikasi algoritma genetika dapat juga digunakan untuk sistem penunjang keputusan cerdas. Di mana sistem ini digunakan untuk mengelola rantai pasokan pada agroindustri hortikultura yang telah diteliti oleh Radityo Andi Dharma dan Yandra Arkeman (2008) dan telah diterbitkan dalam bentuk jurnal. Bab 8 ini akan membahas sekilas tentang penelitian yang telah dilakukan.

8.1 Latar Belakang

Perkembangan teknolgi informasi serta persaingan dunia industri memberikan banyak alternatif bagi konsumen dalam memilih produk, akibatnya tuntutan konsumen menjadi lebih tinggi. Konsumen menuntut antara lain: pelayanan yang lebih cepat, kualitas yang lebih baik, serta harga yang lebih murah. Di era globalisasi saat ini, hal tersebut dapat menjadi hambatan bagi produsen sayuran di Indonesia yang mayoritas belum menerapkan manajemen yang memadai karena mereka akan bersaing dengan produsen sayuran dari manca negara. Hambatan tersebut telah dibuktikan dengan ketidakmampuan produk pertanian Indonesia bersaing dengan produk pertanian impor baik dalam segi harga maupun kualitas. Saat ini persaingan yang sesungguhnya bukanlah persaingan antarperusahaan, melainkan persaingan antar jaringan kerja. Sebuah jaringan yang terdiri dari pemasok, perusahaan,

distributor, dan pengecer, akan saling bekerja sama untuk menghasilkan produk akhir dengan kualitas tinggi, waktu yang lebih singkat, serta harga yang lebih terjangkau. Hanya tim yang memiliki jaringan kerja yang paling efektif dan efisien yang dapat memenangkan persaingan.

Supply Chain Management merupakan mekanisme pengelolaan rantai pasokan untuk mengoptimalkan nilai-nilai yang terdapat disepanjang rantai pasokan dengan cara mengoptimalkan aliran barang, aliran informasi, dan aliran uang di dalam rantai pasokan agar produk yang sampai ke konsumen dapat memberikan kepuasan dalam hal ketepatan waktu pengiriman, kualitas barang, dan harga yang terjangkau, sehingga pada akhirnya akan memberikan keuntungan yang maksimal kepada seluruh anggota yang terlibat dalam rantai pasokan (Chopra dan Meindl 2004, Apaiah dan Hendrix 2004)

Aplikasi sistem ini bertujuan untuk sebagai penunjang keputusan cerdas untuk mengelola rantai pasokan pada agroindustri hortikultura sesuai dengan kebutuhan perusahaan, mengintegrasikan pendekatan-pendekatan pemecahan persoalan *Supply Chain Management* ke dalam sistem penunjang keputusan cerdas (Dhar dan Stein 1997). Sistem penunjang keputusan cerdas ini pun menerapkan Algoritma genetika (Gen dan Cheng 1997, Goldberg 1989, De Jong 2006) sebagai bagian dari sistem untuk memperoleh hasil optimum yang berkualitas dalam waktu yang lebih singkat.

8.2 Formulasi Masalah

Rantai pasokan merupakan mekanisme penyediaan produk sampai kepada konsumen akhir yang melibatkan pemasok, produsen, distributor, dan pengecer. Oleh sebab itu, pengelolaan yang baik atas aliran informasi, aliran barang, dan aliran keuangan yang terjadi di antara anggota rantai pasokan memegang peranan yang sangat penting dalam menentukan efisiensi rantai pasokan. Rantai pasokan yang efisien

akan memberikan kepuasan kepada konsumen dalam hal ketepatan waktu pengiriman, kualitas barang, dan harga yang terjangkau, sehingga pada akhirnya akan memberikan keuntungan yang maksimal kepada seluruh anggota yang terlibat dalam rantai pasokan.

Dalam pelaksanaannya, seorang pengambil keputusan yang mengelola rantai pasokan akan dihadapkan kepada berbagai persoalan yang rumit dan kompleks (Apaiah dan Hendrix 2004) karena pengelolaan rantai pasokan merupakan suatu persoalan yang dinamis dan melibatkan banyak elemen yang saling berkaitan (Brycesson dan Smith 2008). Seorang pengambil keputusan sangat dituntut untuk menghasilkan keputusan yang terbaik dalam waktu yang singkat, sehingga diperlukan suatu alat yang dapat menyediakan informasi yang mendukung proses pengambilan keputusan secara cepat, ringkas, dan informatif (Simch-Levy, Kaminsky dan Simchi-Levy 2000). Alat tersebut dinamai Sistem penunjang keputusan cerdas untuk mengelola rantai pasokan pada agroindustri hortikultura (*Intelligent Decision Support System for Supply Chain Management* atau IDSS-SCM).

8.3 Pengembangan Sistem

IDSS-SCM digunakan untuk mengelola rantai pasokan agroindustri hortikultura dengan mengintegrasikan elemen-elemen yang terdapat dalam *Supply Chain Management* seperti *Forecasting*, *Inventory Management*, *Aggregate Planning*, *Resource Planning*, dan *Transportation Management*. Selain itu IDSS-SCM menggunakan metode penyelesaian persoalan yang diadaptasi dari bidang kecerdasan buatan yaitu Algoritma genetika, sehingga dengan menggunakan sistem ini diharapkan proses pengambilan keputusan yang berkaitan dengan pengelolaan rantai pasokan agroindustri hortikultura dapat berlangsung dengan lebih efektif dan efisien.

8.3.1 Model Peramalan

Pengelolaan rantai pasokan dalam sistem dimulai dengan membuat perkiraan jumlah permintaan dari setiap jenis produk selama satu tahun ke depan berdasarkan hasil panen tahun-tahun sebelumnya. Perkiraan jumlah permintaan ini dihitung di dalam Model Peramalan Permintaan Produk.

Metode peramalan yang digunakan dalam model ini ada 6 macam, yaitu: *Moving Average*, *Double Moving Average*, *Single Exponential Smoothing*, *Adaptive Exponential Smoothing*, *Brown's Method*, dan *Holt's Method* (Heizer dan Render 2004). Hasil peramalan yang dipilih merupakan *output* yang memiliki nilai *Mean Absolute Percentage Error* (MAPE) terkecil dari masing-masing metode. Hasil peramalan tersebut kemudian dimuluskan dengan metode *Seasonal Adjustments*, sehingga hasil peramalan yang diperoleh mengikuti pola musiman berdasarkan pola musim pada tahun-tahun sebelumnya (Turban *et al.* 1991). Data hasil peramalan yang telah dimuluskan kemudian dibagi dengan nilai rendemen produk untuk mendapatkan prediksi kebutuhan bahan baku produk. Untuk prediksi produksi sayur, data hasil peramalan yang telah dimuluskan kemudian dikalikan dengan nilai revisi produksi untuk mendapatkan rencana produksi sayuran yang ideal. Nilai revisi produksi digunakan untuk merevisi prediksi jumlah sayuran yang dihasilkan agar sesuai dengan kapasitas produksi pemasok yang sebenarnya.

8.3.2 Model Rencana Tanam

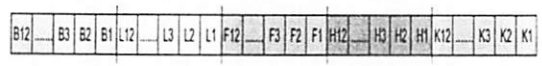
Setelah melakukan sistem peramalan agar bahan baku berupa sayuran dapat dipanen dengan jumlah yang sesuai saat dibutuhkan, perlu dilakukan perencanaan tanam yang meliputi kapan para pemasok mulai menanam sayur, berapa banyak kebutuhan benihnya, dan berapa luas lahan yang dibutuhkan. Perhitungan yang berkaitan dengan rencana tanam ini dilakukan di dalam Model Rencana Tanam. Model ini akan menghitung jumlah sayur yang harus disediakan oleh para

pemasok setiap minggu, kapan pemasok harus mulai menanam sayur, berapa jumlah benih yang dibutuhkan setiap minggunya oleh para pemasok, berapa biaya untuk membeli benih tersebut, dan berapa luas lahan yang harus ditanami/dipanen oleh para pemasok setiap minggunya.

8.3.3 Model Perencanaan Agregat

Pekerja sangat dibutuhkan untuk menghasilkan produk-produk serta berperan dalam proses peningkatan nilai tambah sayuran. Proses yang dilakukan dimulai dari penerimaan, sortasi, pencucian, *trimming*, pemotongan, dan pengepakan sayuran. Oleh karena itu, dibutuhkan pengalokasian sumber daya meliputi penentuan jumlah tenaga kerja harian, berapa banyak tenaga kerja baru yang harus direkrut, berapa banyak tenaga kerja yang diberhentikan, alokasi jam kerja lembur, serta jumlah karyawan borongan yang dibutuhkan. Seluruh perhitungan yang berkaitan dengan alokasi tenaga kerja terdapat dalam Model Perencanaan Agregat. Metode yang digunakan dalam model ini ialah algoritma genetika.

Variabel keputusan yang digunakan dalam model ini direpresentasikan dalam bentuk kromosom, di mana bentuk representasi kromosom dapat dilihat pada Gambar 8.1.



Gambar 8.1 Representasi kromosom dari variabel keputusan fungsi linier perencanaan agregat.

Kromosom pada perencanaan agregat terdiri atas 60 buah gen, yaitu gen K1,K2,K3,...,K12 yang merepresentasikan Jumlah Tenaga Kerja Harian pada bulan ke-1 sampai bulan ke-12, gen H1,H2,H3,...,H12 yang merepresentasikan Jumlah Tenaga Kerja yang direkrut pada bulan ke-1

sampai bulan ke-12, gen F1,F2,F3,...,F12 yang merepresentasikan Jumlah Tenaga Kerja yang di PHK pada bulan ke-1 sampai bulan ke-12, gen L1,L2,L3,...,L12 yang merepresentasikan Jumlah Produksi Lembur pada bulan ke-1 sampai bulan ke-12, dan gen B1,B2,B3,...,B12 yang merepresentasikan Jumlah Produksi Borongan pada bulan ke-1 sampai bulan ke-12.

Variabel keputusan yang terdapat pada kromosom tersebut akan dilakukan fungsi evaluasi. Fungsi evaluasi ini bertujuan untuk meminimumkan total biaya tenaga kerja selama satu tahun ke depan.

Model Perencanaan Agregat adalah sebagai berikut:

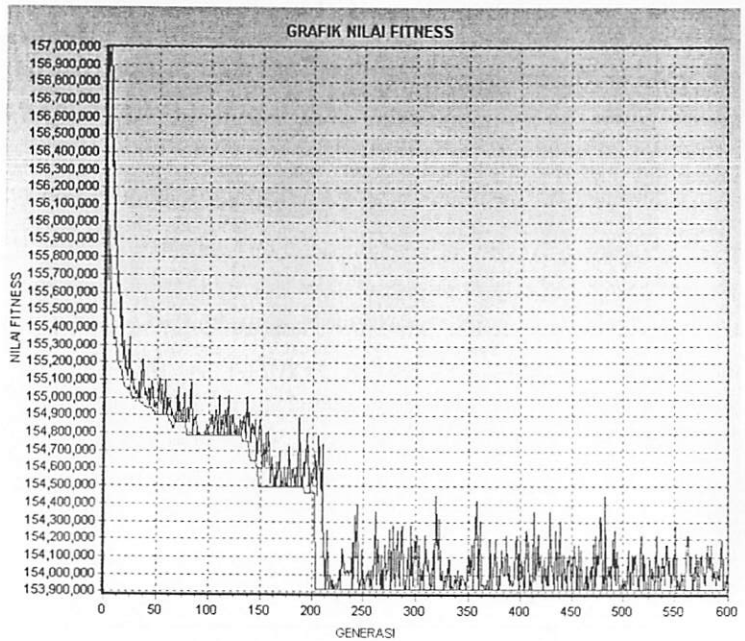
$$TCTK = \sum_{i=1}^{12}(TKH_i \times PRTKB \times URK) + \sum_{i=1}^{12}(TKHR_i \times BR) + \sum_{i=1}^{12}(TKHP_i \times BP) + \sum_{i=1}^{12}(PLB_i \times ULK) + \sum_{i=1}^{12}(PB_i \times UBK)$$

Di mana:

TCTK = *Total Cost* Tenaga Kerja
TKH_i = Jumlah Tenaga Kerja Harian bulan ke-*i*
PRTKB = Kapasitas Produksi Reguler Tenaga Kerja per bulan
URK = Upah Reguler per Kilogram produk yang dihasilkan
$TKHR_i$ = Jumlah Tenaga Kerja Harian yang di Rekrut pada bulan ke-*i*
BR = Biaya Rekrut Tenaga Kerja Harian per orang
$TKHP_i$ = Jumlah Tenaga Kerja Harian yang di PHK pada bulan ke-*i*
BP = Biaya Pemutusan Hubungan Kerja per orang
PLB_i = Produksi Lembur bulan ke-*i*
ULK = Upah Lembur per Kilogram Produk yang dihasilkan
PB_i = Produksi Borongan bulan ke-*i*
UBK = Upah Borongan per Kilogram produk yang dihasilkan

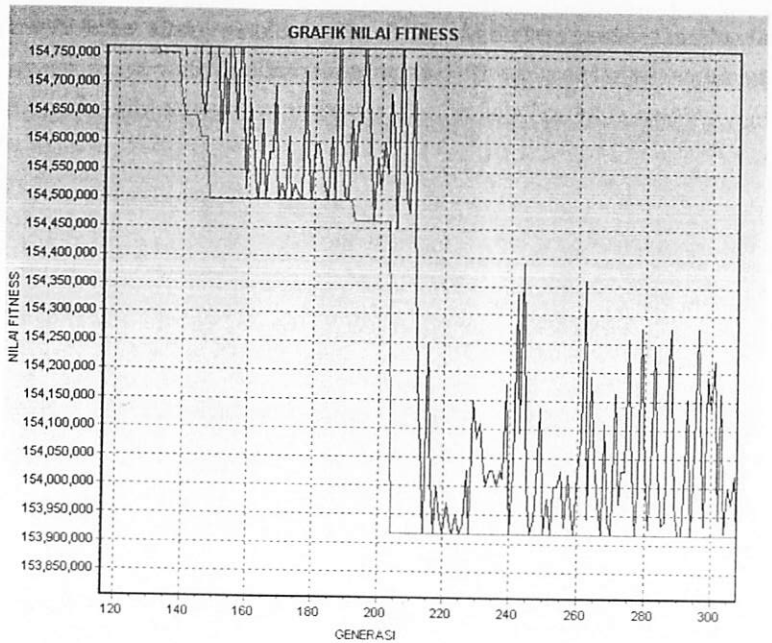
Hasil eksekusi Model Perencanaan Agregat disajikan dalam 2 buah grafik. Gambar 8.2 menunjukkan grafik nilai *fitness* rata-rata dan nilai *fitness* minimum dari generasi pertama sampai generasi

maksimum, sedangkan Gambar 8.3 menunjukkan grafik nilai *fitness* minimum dari generasi ke-120 sampai generasi ke-300.



Gambar 8.2 Grafik nilai *fitness* rata-rata dan nilai *fitness* minimum dari tiap generasi

Nilai *fitness* minimum konvergen sejak generasi ke-204 sampai generasi ke-600. Keadaan tersebut terjadi karena meskipun proses penyilangan dan mutasi dilakukan sepanjang generasi tersebut, tetapi proses tersebut tidak lagi dapat memunculkan kromosom-kromosom yang lebih baik, sehingga dapat disimpulkan bahwa solusi optimal telah tercapai pada nilai TCTK= 153915400.



Gambar 8.3 Grafik nilai *fitness* minimum dari Generasi ke-120 sampai ke-300

8.3.4 Model *Material Requirement Planning* (MRP)

Model ini berfungsi untuk menghitung kebutuhan bahan baku kemasan dari setiap jenis produk berdasarkan prediksi permintaan produk tersebut selama satu tahun ke depan dengan biaya yang dibutuhkan (*Bills of Materia)l* dari masing-masing produk. Dalam sistem yang dikembangkan menggunakan 2 model MRP, yaitu MRP I dan MRP II.

Model *Material Requirements Planning* I digunakan untuk menghitung prediksi kebutuhan material kemasan setiap jenis produk sayuran setiap bulan selama satu tahun ke depan. Tujuannya adalah membantu divisi pengemasan untuk merencanakan kebutuhan kemasan

setiap jenis produk pada bulan-bulan berikutnya dengan lebih baik, sehingga material kemasan selalu tersedia dalam jumlah yang tepat saat dibutuhkan. Data yang digunakan sebagai masukan untuk model ini adalah hasil prediksi permintaan setiap jenis produk yang diambil dari basis data, biaya yang dibutuhkan masing-masing produk, unit pembelian, dan berat per unit pembelian. *Output* Model *Material Requirements Planning* I selanjutnya disimpan ke *database* dan digunakan sebagai *input* untuk Model *Material Requirements Planning* II.

Model *Material Requirements Planning* II digunakan untuk menghitung kebutuhan material kemasan dari seluruh jenis produk sayuran setiap bulan selama satu tahun ke depan serta biaya yang dibutuhkan untuk membeli material kemasan. Informasi yang dihasilkan dari model ini berguna bagi divisi pengemasan dan divisi pengadaan, agar divisi pengemasan dapat mengajukan rencana pembelian bahan baku kemasan kepada divisi pengadaan dengan waktu yang lebih singkat dan informasi yang lebih terperinci, sehingga material kemasan selalu tersedia dalam jumlah yang tepat saat dibutuhkan dan dapat membantu kelancaran proses produksi.

Data masukan yang dibutuhkan dalam model ini adalah hasil prediksi kebutuhan material kemasan dari setiap produk selama satu tahun ke depan. Data-data tersebut kemudian diagregasi, sehingga total kebutuhan setiap jenis material kemasan selama satu tahun dapat diketahui. Data masukan lainnya adalah daftar seluruh material kemasan yang digunakan oleh divisi pengemasan, satuan material kemasan, dan harga per satuan material kemasan.

Hasil perhitungan menunjukkan bahwa besarnya biaya material kemasan berbanding lurus dengan jumlah permintaan produk. Semakin tinggi permintaan suatu produk, material kemasan yang dibutuhkan semakin banyak.

8.3.5 Model Manajemen *Inventory* Kemasan

Model Manajemen *Inventory* Kemasan berfungsi untuk mengelola persediaan material kemasan selama satu tahun ke depan agar tidak terjadi *outstock* persediaan yang dapat mengganggu kelancaran proses produksi. Model ini juga bertujuan membantu manajer divisi pengemasan dalam menentukan kebijakan-kebijakan yang terkait dengan pengadaan dan pembelian material kemasan, seperti jumlah pembelian optimal dalam setiap *order* untuk meminimumkan biaya pemesanan dan biaya penyimpanan, jumlah pembelian dalam satu tahun, rentang waktu pembelian antara satu *order* dengan *order* berikutnya, dan perkiraan biaya yang dibutuhkan untuk pengadaan material kemasan tersebut.

8.3.6 Model Rute Pengiriman

Model Rute Pengiriman digunakan untuk menentukan rute pengiriman terpendek bagi truk-truk distribusi yang mengantarkan produk sayuran ke konsumen. Tujuannya adalah meminimumkan biaya pemakaian bahan bakar solar dan meminimumkan waktu pengiriman.

Persoalan penentuan rute pengiriman terpendek merupakan persoalan *Traveling Salesman Problem* (TSP) yang bersifat *NP-Hard* (*Non-deterministic Poynomial*), artinya waktu yang dibutuhkan untuk menyelesaikan persoalan ini akan semakin meningkat secara eksponensial seiring dengan bertambahnya jumlah elemen dalam persoalan tersebut (Bagchi 1999).

Persoalan penentuan rute pengiriman terpendek di optimasi menggunakan Algoritma genetika. Fungsi *fitness* dalam Algoritma genetika adalah total jarak tempuh rute pengiriman dalam Model Rute Pengiriman sebagai berikut:

$$TJRK = \left(\sum_{i=1}^{n-1} JRK(\text{Lokasi } X_i, \text{Lokasi } X_{i+1}) \right) + JRK(\text{Lokasi } X_1, \text{Lokasi } X_n)$$

Dengan syarat:

(Lokasi $X_i \neq$ Lokasi X_{i+1}) dan (Lokasi $X_1 \neq$ Lokasi X_n)

Di mana:

TJRK	=	Total jarak tempuh
n	=	Jumlah lokasi pengiriman
X	=	Himpunan bilangan dengan kombinasi $n!$
i	=	Bilangan ke-i pada himpunan X
Lokasi X_i	=	Menunjukkan Lokasi ke-X

Misalnya:

Jika X={2,3,1} dan i=2, maka Lokasi X_i = Lokasi 3

JRK(Lokasi X_iLokasi X_{i+1}) = Jarak antara Lokasi X_i dengan Lokasi X_{i+1}

Representasi kromosom yang digunakan dalam Model Rute Pengiriman adalah *path representation chromosome*, di mana setiap gen dalam kromosom memiliki sebuah nilai yang merepresentasikan sebuah lokasi pengiriman (Goldberg 1989). Setiap gen dalam kromosom dilarang memiliki nilai yang sama. Gambar 8.4 adalah contoh representasi kromosom untuk persoalan TSP dengan 10 lokasi pengiriman.

5	3	10	7	4	1	9	6	8	2

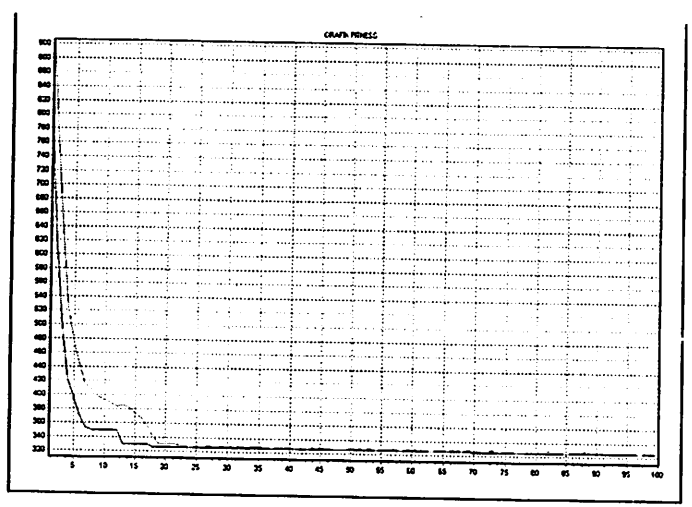
Gambar 8.4 Representasi kromosom untuk menyelesaikan persoalan TSP pada Model Rute Pengiriman

Pada contoh di atas, kromosom tersebut merepresentasikan rute pengiriman dengan urutan pengiriman: Lokasi 5-Lokasi 3-Lokasi 10-Lokasi 7-Lokasi 4-Lokasi 1-Lokasi 9-Lokasi 6-Lokasi 8-Lokasi 2. Panjang

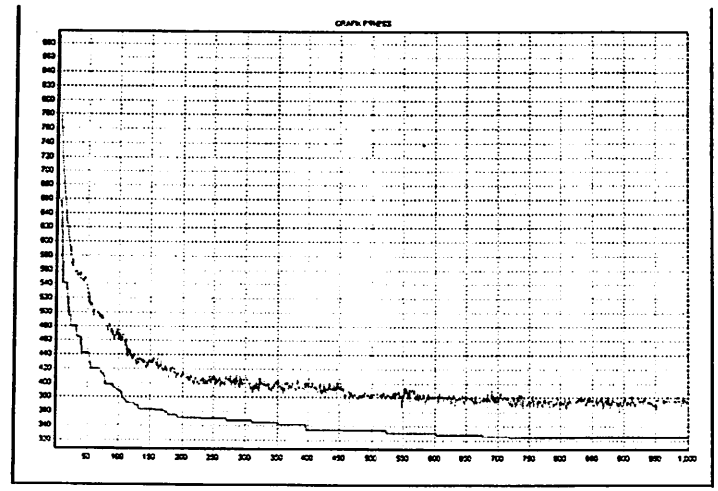
kromosom bergantung pada banyaknya jumlah lokasi pengiriman. Pada contoh di atas, terdapat 10 lokasi pengiriman, sehingga panjang kromosomnya terdiri atas 10 gen.

Hasil eksekusi Model Rute Pengiriman disajikan dalam 2 buah grafik. Gambar 8.5 menunjukkan grafik nilai *fitness* rata-rata dan nilai *fitness* minimum dari generasi pertama sampai generasi maksimum dengan metode *Edge Recombination Crossover* (ERX). Sedangkan Gambar 8.6 menunjukkan grafik nilai *fitness* rata-rata dan nilai *fitness* minimum dari generasi pertama sampai generasi maksimum dengan metode *Partially Match Crossover* (PMX).

Metode ERX dan PMX merupakan proses pindah silang yang terdapat dalam GA. Proses pindah silang sama halnya dengan proses seleksi yaitu mengambil nilai acak sederhana. Pindah silang merupakan komponen paling penting dalam GA pada proses genetik (Gen dan Cheng 1997).



Gambar 8.5 Grafik nilai *fitness* rata-rata dan minimum dari tiap generasi dengan metode ERX



Gambar 8.6 Grafik nilai *fitness* rata-rata dan minimum dari tiap generasi dengan metode PMX

Berdasarkan verifikasi yang dilakukan terhadap metode penyilangan ERX dan PMX dengan parameter Algoritma genetika yang sama, yaitu ukuran populasi=96, jumlah kromosom induk=32, dan peluang mutasi=5%, diperoleh hasil sebagai berikut:

- Dengan jumlah generasi maksimum =100, metode penyilangan ERX mampu mencapai solusi optimum pada generasi ke-70 yaitu sebesar 325 km. Total waktu yang dibutuhkan untuk eksekusi Algoritma genetika adalah 25 detik dengan kecepatan evaluasi nilai *fitness* per kromosom sebesar 0,0026042 detik.
- Dengan jumlah generasi maksimum =1000, metode penyilangan PMX mampu mencapai solusi optimum pada generasi ke-718 yaitu sebesar 325 km. Total waktu yang dibutuhkan untuk eksekusi Algoritma genetika adalah 2 menit 10 detik dengan kecepatan evaluasi nilai *fitness* per kromosom sebesar 0,0013542 detik.

Metode PMX memiliki dua titik potong secara random yang akan ditukar dengan induk lainnya. Pertukaran antar induk akan menghasilkan keturunan yang mempertahankan urutan dan posisi sel induk (Michalewicz 1996). Sehingga terbukti kecepatan metode PMX lebih cepat 1,9 kali dibandingkan metode ERX. Akan tetapi, jumlah generasi yang dibutuhkan PMX untuk menemukan solusi optimum jauh lebih besar, sehingga total waktu yang dibutuhkan lebih lama dibandingkan dengan metode ERX. Oleh sebab itu dapat disimpulkan bahwa metode penyilangan ERX lebih efisien dalam menemukan solusi optimal dibandingkan dengan metode PMX.

8.4 Kesimpulan

IDSS_SCM dapat membantu pengguna untuk mengelola rantai pasokan pada agroindustri hortikultura selama rentang waktu satu tahun ke depan. Penerapan algoritma genetika dalam IDSS-SCM membantu sistem untuk menghasilkan informasi yang berkualitas dalam waktu yang relatif singkat.

Berdasarkan verifikasi diketahui bahwa metode penyilangan kromosom yang lebih efisien untuk menyelesaikan persoalan *Traveling Salesman Problem* pada model rute pengiriman adalah *Edge Recombination Crossover* (ERX).

Bab 9
Bermain-main dengan Algoritma Genetika

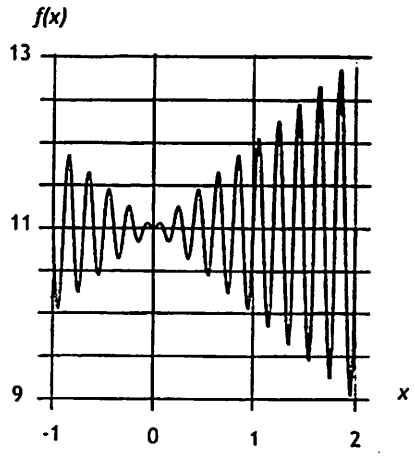
Bab 9 ini secara khusus akan mengajak pembaca untuk bermain-main dengan algoritma genetika. Untuk itu, kami telah menyiapkan sebuah permainan, yaitu berupa implementasi pemecahan fungsi sulit dengan menggunakan beragam nilai parameter algoritma genetika. Kita dapat dengan sebebas mungkin mengatur *setting* awal untuk algoritma genetika, yaitu nilai peluang penyilangan, peluang mutasi, jumlah populasi, dan nilai *seed random number*. Ayo kita coba.

9.1 Permainan Optimasi Fungsi Sulit

Dalam permainan ini, fungsi sulit yang akan dioptimasi adalah :

$$f(x) = x.\sin(10\pi.x) + 11; \quad x = [-1..2];$$

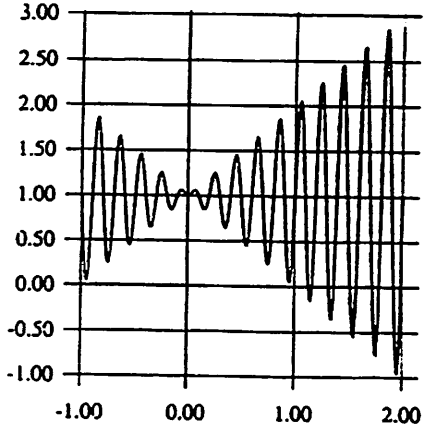
Apabila dituangkan dalam bentuk grafis, fungsi diatas dapat dilihat dalam bentuk seperti gambar di bawah ini.



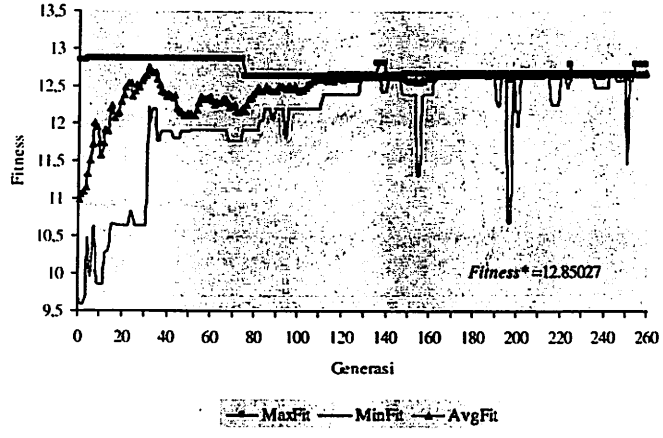
Fungsi sulit di atas merupakan hasil modifikasi terhadap fungsi sulit yang telah digunakan oleh Michalewicz (1996). Dalam eksperimennya, Michalewicz menggunakan fungsi :

$$f(x) = x.\sin(10\pi.x) + 1; \quad x = [-1..2]$$

Apabila dituangkan dalam bentuk grafis, bentuk fungsi adalah seperti gambar di bawah ini.



Terhadap fungsi sulit yang telah dimodifikasi, penulis melakukan eksperimen menggunakan bahasa pemrograman Pascal, dimana hasil peneitian Michalewicz (1996) digunakan sebagai pembanding (*benchmark*). Dalam proses *running* program, penulis menggunakan populasi sejumlah 40, panjang kromosom 22, maksimum generrasi 1.000, peluang penyilangan 0,8, peluang mutasi 0,01, dan *seed random number* sebesar 0,7. Adapun hasil *running* programnya, penulis tuangkan dalam bentuk grafik seperti ppada Gambar 9.1 di bawah ini.



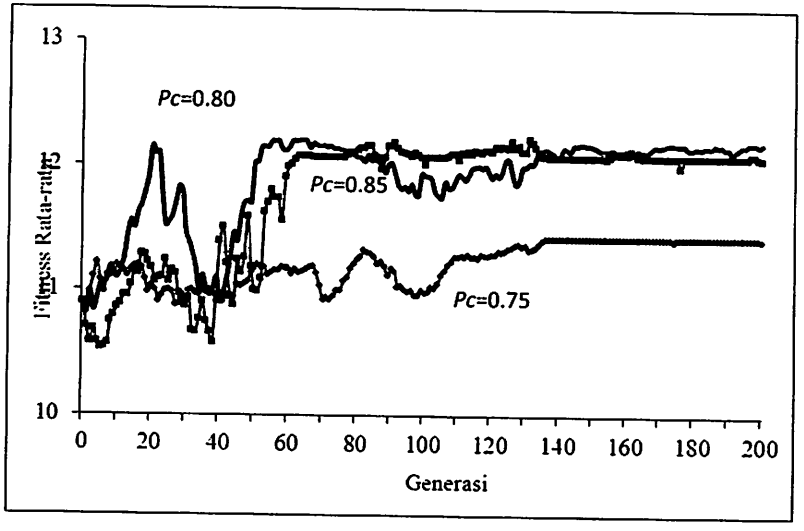
Gambar 9.1 Grafik nilai *fitness* maksimum, *fitness* rata-rata, dan *fitness* minimum untuk 260 generasi pertama (*PopSize=40; LChrom=22 ; MaxGen=1000; Pc=0,80 ; Pm=0,01*)

Dari Gambar 9.1 terlihat bahwa nilai *fitness* rata-rata populasi berangsur-angsur meningkat hingga mencapai puncaknya pada generasi ke-30. Pada saat yang bersamaan, nilai minimum populasi pun kian meningkat. Sementara itu, nilai maksimum populasi sudah ditemukan pada generasi-generasi awal dengan nilai 12,85027. Nilai ini kemudian akan konvergen hingga ke generasi 80. Namun, pada generasi berikutnya, nilai *fitness* maksimum akan menurun seiring dengan

adanya proses mutasi. Meskipun demikian, secara umum nilai *fitness* rata-rata populasi akan pulih dan meningkat kembali.

9.1.1 Variasi Peluang Penyilangan (*Pc*)

Dalam eksperimen berikut ini, kami menggunakan tiga nilai parameter peluang penyilangan (*Pc*) yang berbeda, yaitu masing-masing 0,75, 0,80, dan 0,85. Hasil ekperimennya kami plot seperti pada Gambar 9.2.

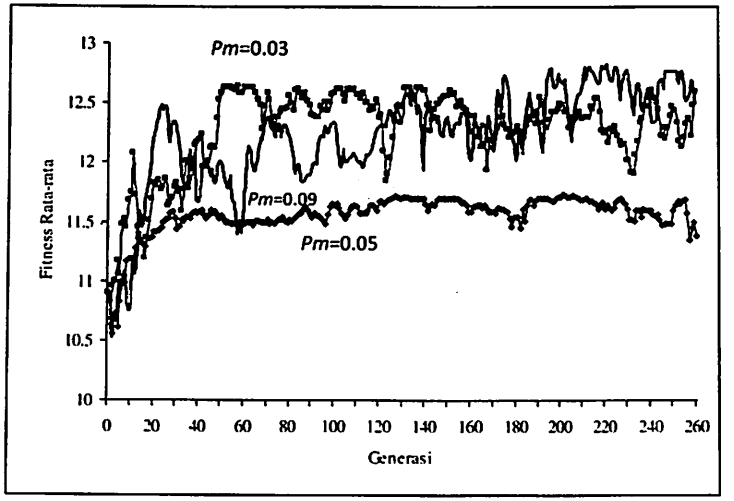


Gambar 9.2 Grafik nilai *fitness* rata-rata dengan variasi nilai peluang penyilangan (*Pc*)

Dari grafik tersebut, terlihat bahwa variasi nilai *Pc* memberikan hasil yang berbeda. Nilai *Pc* 0,80 dan 0,85 memberikan hasil yang lebih baik dibanding *Pc* 0,75. Hal ini menunjukkan bahwa dalam masalah yang dihadapi, dibutuhkan peluang mutasi yang cukup tinggi. Oleh karena penyilangan adalah operator primer, peluang penyilangan dalam populasi pun selayaknya tinggi agar didapat solusi optimal.

9.1.2 Variasi Peluang Mutasi (*Pm*)

Dalam eksperimen berikutnya, kami menggunakan tiga nilai parameter peluang mutasi (*Pm*) yang berbeda, yaitu masing-masing 0,03, 0,09, dan 0,05. Hasil ekperimennya dapat dilihat pada Gambar 9.3.

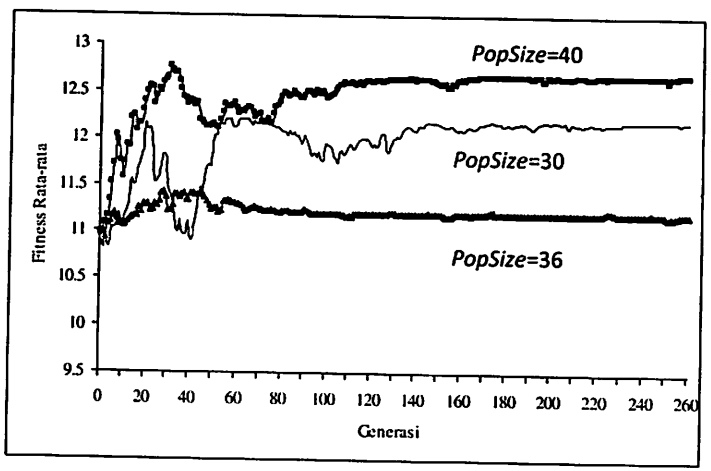


Gambar 9.3 Grafik nilai *fitness* rata-rata dengan variasi nilai peluang mutasi (*Pm*)

Dari grafik tersebut, terlihat bahwa variasi nilai *Pm* memberikan hasil yang berbeda-beda. Nilai *Pm* 0,09 cenderung memberikan hasil yang lebih baik dibanding *Pm* 0,03 dan *Pm* 0,05. Hal ini menunjukkan bahwa dalam masalah yang dihadapi, tingkat peluang 0,09 cukup mendukung operator primer untuk menemukan solusi optimum. Oleh karena mutasi adalah operator sekunder, dan juga sifatnya eksperimental, peluang mutasi dalam populasi pun selayaknya rendah agar algoritma genetika tidak terjebak pada solusi sub-optimum (subkonvergen).

9.1.3 Variasi Jumlah Populasi (*PopSize*)

Dalam eksperimen berikutnya, kami menggunakan tiga nilai parameter jumlah populasi (*PopSize*) yang berbeda, yaitu masing-masing 30, 36, dan 40. Hasil ekperimennya dapat dilihat pada Gambar 9.4.



Gambar 9.4 Grafik nilai *fitness* rata-rata dengan variasi nilai jumlah populasi (*PopSize*)

Dari grafik tersebut, terlihat bahwa variasi jumlah populasi pun memberikan hasil yang berbeda-beda. Jumlah populasi 40 memberikan hasil yang lebih baik dibandingkan dengan jumlah populasi 30 dan 36. Dengan jumlah populasi 40 ini, nilai *fitness* rata-rata populasi meningkat dengan cepat hingga generasi ke-40. Dengan jumlah populasi yang lebih banyak, peluang munculnya heterogenitas pun lebih besar. Artinya, algoritma genetika melakukan pencarian dengan jangkauan yang lebih luas dalam *search space*. Dengan jangkauan yang lebih luas, peluang mendapatkan solusi optimum pun lebih besar.

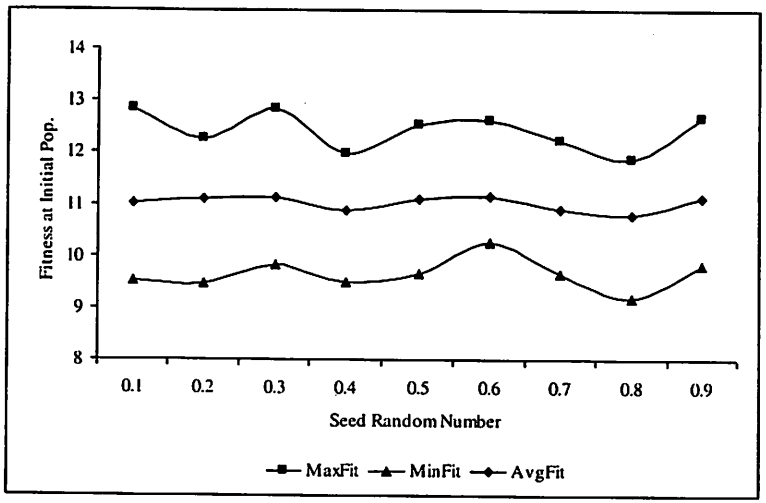
9.1.4 Variasi *Seed Random Number*

Dalam eksperimen kali ini, kami menerapkan hal yang tidak biasa yaitu variasi dalam nilai *seed random number*. Seperti diketahui, *seed random number* adalah nomor yang diinisiasi oleh *user* (manusia) terhadap komputer agar komputer dapat membangkitkan bilangan acak (*pseudo random number*). *Seed random number* ini diperlukan karena dalam eksperimen, manusia lah yang melakukan aksi, sementara komputer akan memberikan rekasi.

Nilai *seed random number* yang digunakan adalah seperti pada tabel di bawah ini. Hasil ekperimennya kami fokuskan pada populasi awal (*initial population*) yaitu nilai *fitness* maksimum, rata-rata, dan minimum. Hasil eksperimen ini dapat dilihat pada Gambar 9.5.

Seed Random	MaxFit	MinFit	AvgFit
0,1	12,850273	9,529937	11,014890
0,2	12,250105	9,454296	11,094387
0,3	12,850258	9,823600	11,130066
0,4	11,986807	9,495945	10,880304
0,5	12,537702	9,656364	11,088255
0,6	12,618558	10,266702	11,151170
0,7	12,223162	9,665914	10,896281
0,8	11,860265	9,184028	10,790403
0,9	12,663651	9,820403	11,137673

Dari hasil eksperimen, terlihat ada beberapa kemiripan hasil meski *seed random number*-nya berbeda. Kemiripan ini terlihat dari adanya kesamaan nilai *fitness* antara *seed random number* 0,1, 0,3, dengan 0,9. Kemudian, kesamaan pula terjadi antara *seed random number* 0,2 dan 0,5. Meskipun demikian, perlu ditekankan bahwa nilai *seed random* tidak berpengaruh banyak terhadap hasil eksperimen. Pengaruhnya hanya pada *initial population* saja. Untuk populasi berikutnya, algoritma genetikalah yang akan menentukan.



Gambar 9.5 Grafik nilai *fitness* dengan variasi nilai *seed random number*

Dari eksperimen terhadap *seed random number* ini, dapat diambil sebuah premis bahwa proses evolusi, seleksi, dan mutasi yang terjadi di alam adalah proses acak alami. Proses ini berlangsung sejak terciptanya alam semesta beserta isinya. Keacakan inilah yang kemudian dicoba diadopsi oleh manusia (dalam program komputer) menggunakan *pseudo random number*. Oleh karena namanya saja "*pseudo*" (samar), dalam komputer tidak ada bilangan yang benar-benar acak. Meskipun begitu, keacakan yang dimiliki komputer ini sudah dapat memenuhi kaidah manusia untuk dapat disebut sebagai bilangan acak.

9.2 Sekilas Tentang *Random Number*

Sistem ataupun aplikasi seperti simulasi biasanya menggunakan bilangan acak (*random number*). Akan tetapi dalam kenyataannya, komputer tidak dapat membuat bilangan acak, di mana bilangan acak merupakan bilangan yang muncul berdasarkan peluang. Oleh karena itu, berkembanglah algoritma dasar untuk membuat bilangan acak seperti

contohnya dalam teknik *pseudo random number generation* (PRNG) yang menggunakan metode *linear congruential*.

Sebagai contoh (acm.uva.es/p/v3/350.html), jika L merupakan nilai terakhir yang dihasilkan, nilai selanjutnya ialah $(Z \times L + I) \bmod M$, di mana Z adalah konstanta perkalian, I adalah kenaikan nilai, dan M adalah konstanta pembagi modulu.

Z = 7, I = 5, dan M = 12. Jika angka random pertama (*seed*) adalah L = 4 , kita bisa mendapatkan nilai selanjut dengan cara:

Angka terakhir (L)	$(Z \times L + I)$	Angka selanjutnya $(Z \times L + I) \bmod M$
4	33	9
9	68	8
8	61	1
1	12	0
0	5	5

Bilangan acak yang dihasilkan PNRG yang mengandung *noise* dapat digunakan melalui www.random.org beserta penjelasannya.

Dalam algoritma genetika, bilangan acak sering digunakan untuk menentukan inisialisasi populasi. Bilangan acak dapat dibentuk menggunakan sintak *random* yang terdapat di fungsi delphi atau *rand()* yang biasanya terdapat pada excel, MATLAB, php. Sintaks bilangan acak biasanya bernilai dari 0 sampai dengan 1. Dari sintaks tersebut kita dapat membentuk bilangan acak sesuai dengan kebutuhan. Seperti contoh dalam delphi:

```
var
  float : single;
  int   : Integer;
  i     : Integer;
begin
//menghasilkan bilangan acak range 0 - 1
   for i := 1 to 3 do
  begin
    float := Random;

end

// menghasilkan bilangan acak dengan range 0 - 99
  for i := 1 to 3 do
  begin
    int := 1 + Random(100);
    ShowMessage('int = '+IntToStr(int));
  end;
end;
```

Contoh di atas dapat digunakan untuk inisialisasi populasi jika alele dalam kromosom menggunakan bilangan real. Jika kita menggunakan bilangan *binary bit* yaitu 0 dan 1 bisa menggunakan algoritma seperti di bawah ini:

```
Input p = 0.5;
If rand() < p
     Then x = 0;
Else
     x = 1;
end
```

Bilangan acak yang dibentuk dapat dikembangkan selanjutnya dalam fungsi genetik seperti seleksi dalam menentukan peluang terpilihnya orang tua, pindah silang dalam hal menentukan titik potong, dan mutasi. Penggunaan bilangan acak tersebutlah yang begitu menarik dalam memecahkan masalah menggunakan algoritma genetika.

Bab 10
Penutup

Di awal dan di sepanjang buku ini telah diperlihatkan bagaimana algoritma genetika dapat digunakan untuk membantu manajer dalam pengambilan keputusan, terutama dalam menghadapi permasalahan yang sulit (*complex*), tidak terstruktur dengan baik (*ill-structured*) dan berskala besar (*large-scale*). Walaupun sudah cukup banyak dikaji dan diterapkan, peluang-peluang inovasi algoritma genetika untuk bisnis dan industri masih sangat terbuka lebar. Inovasi baru dapat dilakukan dengan menemukan struktur kromosom dan operator reproduksi yang lebih efisien dan serba guna dengan tujuan untuk memperbaiki struktur kromosom dan operator reproduksi yang sudah ada selama ini. Bersamaan dengan itu, inovasi dapat juga dilakukan dengan menemukan bidang-bidang aplikasi baru untuk algoritma genetika. Dengan ditemukannya bidang-bidang aplikasi baru ini, keampuhan algoritma genetika akan semakin terbukti dan usaha manusia untuk mewujudkan "mesin pintar" idamannya akan semakin lama semakin terlihat keberhasilannya.

10.1 Peluang-peluang Inovasi Algoritma Genetika dalam Bisnis dan Industri

Di bawah ini akan diuraikan beberapa peluang inovasi di bidang algoritma genetika untuk bisnis dan industri, antara lain sebagai berikut:

- Sebuah bank dapat menggunakan algoritma genetika untuk melakukan penyaringan (*screening*) dari sekian banyak aplikasi peminjaman uang. Seperti yang kita diketahui bahwa banyak bank

yang mempunyai program untuk meminjamkan uang kepada pelanggannya. Untuk mempermudah pekerjaannya, bank tersebut dapat membuat suatu sistem komputer yang mampu mengidentifikasi calon peminjam dengan risiko kemacetan pengembalian uang yang tinggi, untuk selanjutnya dianalisis dengan lebih seksama oleh analis kredit. Sistem komputer tersebut harus mampu memberikan identifikasi, sisi mana dari calon peminjam yang harus diteliti lebih lanjut oleh analis kredit.

- Sebuah perusahaan penjual data (*data vendor*) dapat membuat algoritma genetika yang memungkinkan konsumennya untuk meminta data dengan cara yang lebih natural. Misalnya, perusahaan ini dapat membuat suatu sistem komputer yang memungkinkan konsumennya untuk mencari data tentang produk yang hampir sama kualitasnya dengan produk X yang sudah terkenal dengan area penjualan di dekat kota Y.
- Seorang investor dapat menggunakan algoritma genetika untuk memonitor peluang investasi. Oleh karena investor ini berhadapan dengan banyak usaha yang baru muncul, data tentang aset dan pasar dari usaha ini sangat terbatas. Untuk itu perlu suatu sistem yang mampu meramalkan peluang investasi berdasarkan data, model, dan pengetahuan yang didapatkan dari para ahli ekonomi dan sumber lain.
- Sebuah perusahaan transportasi dapat membuat algoritma genetika untuk menentukan rute pengiriman barang yang meminimumkan waktu dan biaya pengiriman barang untuk seluruh tempat tujuannya.
- Industri manufaktur yang memproduksi berbagai jenis komputer sesuai dengan selera pemesan, dapat menggunakan algoritma genetika untuk menentukan konfigurasi komponen yang paling baik untuk jenis komputer yang dipesan.

- Sebuah perusahaan eksportir dapat membuat algoritma genetika yang bisa meramalkan nilai tukar uang (valuta asing) di masa yang akan datang sehingga perusahaan ini dapat mengambil keuntungan yang lebih besar dari kontrak-kontrak yang dibuatnya. Perusahaan ini dapat menggunakan data historis tentang bentuk kurva nilai tukar uang untuk sistem yang akan dibuat. Dengan adanya *Intelligent Business Support* (IBS) ini perusahaan berharap bisa mendapatkan pola pergerakan kurva sehingga nilai tukar uang di masa datang dapat diramalkan dengan tepat.
- Sebuah bisnis properti dapat menggunakan algoritma genetika untuk menentukan nilai properti berdasarkan fitur-fitur yang ada pada properti tersebut. Data-data properti masa lampau yang telah dijual beserta karakteristik-karakteristiknya dapat digunakan untuk keperluan di atas. Dengan sistem informasi yang dilengkapi dengan algoritma genetika, bisnis properti ini akan dapat melakukan pendugaan secara cepat (*quick estimate*) terhadap nilai properti baru.
- Sebuah sekolah bisnis dapat membuat algoritma genetika untuk meramalkan karakteristik mahasiswanya. Model karakteristik mahasiswa ini dapat ditentukan berdasarkan data historis dari calon mahasiswa, mahasiswa yang sedang belajar, dan alumninya selama beberapa tahun terakhir. Selanjutnya data tentang karakteristik mahasiswa ini dapat dipakai untuk menjaring mahasiswa baru yang potensial. Dengan menggunakan sistem informasi bisnis yang dilengkapi dengan algoritma genetika ini biaya promosi dan *recruitment* mahasiswa baru dapat diturunkan dan jumlah serta mutu mahasiswa yang masuk bisa ditingkatkan.

Seperti telah diuraikan di sepanjang buku ini, kelebihan utama dari teknik-teknik "*artificial intelligence*" adalah kemampuannya untuk

meniru proses belajar dan berpikir manusia. Dengan kelebihan ini para ahli yakin bahwa di masa-masa akan datang umat manusia di dunia ini akan didampingi oleh berbagai macam mesin cerdas untuk mempermudah hidup dan menggantikan fungsi manusia. Seorang pilot dapat tidur dengan nyenyak sementara penerbangan dikendalikan oleh *auto pilot* yang cerdas. Seorang dokter tidak perlu bekerja sampai larut malam, karena sebagian tugasnya telah dikerjakan oleh sebuah sistem diagnosis yang handal berbasiskan algoritma genetika. Seorang manajer dapat bermain dan berkumpul dengan keluarganya sementara jalannya perusahaan diatur oleh sebuah sistem penunjang keputusan bisnis yang cerdas berbasiskan algoritma genetika, yang dapat membuat keputusan dengan cara belajar dan berpikir sama seperti (atau bahkan melebihi?) yang dia lakukan. *Why not*?

10.2 Tokoh-tokoh Algoritma Genetika

Seiring dengan perkembangan inovasi algoritma genetika, para peneliti maupun tokoh-tokoh yang ikut andil kian bertambah. Berikut ini adalah tokoh-tokoh algoritma genetika yang sering aktif dalam riset, seminar, maupun publikasi ilmiah internasional.

a. John Holland

John Holland, 83 tahun (lahir 2 Februari 1929) dikenal sebagai orang yang pertama kali mengem-bangkan ilmu algoritma genetika pada tahun 1960-an di University of Michigan. Kala itu, ia bersama rekan-rekannya mencoba membuat simulasi sistem biologis dan proses evolusi alami dalam sebuah program komputer. Hasil eksperimen tersebut memperlihatkan kalau ternyata sistem biologis dan proses evolusi memiliki kemiripan dalam "fungsi optimasi" masalah pencarian

Holland mendapatkan gelar kesarjanaannya dari MIT tahun 1950. Semenjak lulus dari sana, ia pertamakali bergabung di perusahaan komputer IBM dari tahun 1950 sampai 1952. Kemudian, ia mendapatkan gelar M.A (bidang matematik) dan gelar PhD (bidang ilmu komunikasi) dari University of Michigan.

Semenjak lulus dari University of Michigan, Holland berkarier sebagai dosen di almamaternya dan menjadi profesor penuh di Departemen Psikologi. Sebagai akademisi, Holland aktif berdiskusi dengan kolega-kolega yang mempunyai visi dan pemikiran yang sama. Dari diskusi-diskusi itu, terbentuklah sebuah perkumpulan BACH, dengan anggotanya Arthur Burks (ahli komputer), Robert Axelrod (ahli politik dan penulis buku The Evolution of Cooperation), Michael Cohen (ahli politik dan organisasi), dan John Holand sendiri.

Selain di University of Michigan, Holland juga pernah bergabung dengan Santa Fe Institute sebagai steering committee. Di sana, Holland aktif mempromosikan ilmu computer modeling untuk pemecahan masalah-masalah komplek. Ia juga kerap bertukar pikiran dengan tokoh-tokoh ternama seperti Brian Arthur (seorang ekonom) dan Murray Gell-Mann (peraih Nobel fisika). Tahun 1993, ia menerima hibah dana MacArthur Grant, sebuah hibah yang cukup prestisius di Amerika.

Sumber:

- http://www.lsa.umich.edu/psych/people/ (diakses pada 23 September 2012)
- http://www.eecs.umich.edu (diakses pada 23 September 2012)
- http://masi.cscs.lsa.umich.edu (diakses pada 23 September 2012)
- http://en.wikipedia.org/wiki/John_Henry_Holland (diakses pada 23 September 2012)
- http://library.thinkquest.org (dikases pada 23 September 2012)
- http://www.scholarpedia.org/article/user:John_H._Holland (diakses pd 23 September 2012)

b. David E. Goldberg

David E. Goldberg adalah salah seorang murid John Holland yang kini menjabat sebagai Direktur Illinois Genetic Algorithms Laboratory (IlliGAL), sebuah pusat penelitian dan pengembangan algoritma genetika di Universitas Illinois, Amerika. Goldberg mendapatkan gelar kesarjanaannya (BSE) di bidang teknik sipil tahun 1975, gelar master (MSE) tahun 1976, dan gelar doktor (PhD) tahun 1983 dari universitas yang sama, Universitas Michigan.

Selama tahun 1976 sampai 1980, ia telah menduduki posisi penting di perusahaan Stoner Associates of Carlisle, sebagai *Project Engineer* dan *Marketing Manager*. Pada tahun 1984, ia pindah ke Universitas Alabama di Tuscaloosa. Kemudian tahun 1990, Goldberg pindah ke Universitas Illinois.

Beberapa penghargaan bergengsi telah diterima oleh Goldberg. Dua diantaranya yaitu *U.S. National Science Foundation Presidential Young Investigator Award* (tahun 1985) dan *Associate of the Center for Advanced Study* (dari Universitas Illinois, tahun 1995). Selain itu, Goldberg juga menjadi pendiri ISGEC (*International Society for Genetic and Evolutionary Computation*), sebuah perhimpunan tingkat internasional yang melakukan pengembangan ilmu algoritma genetika dan komputasi evolusioner untuk diaplikasikan di dunia nyata.

Sumber:

- http://en.wikipedia.org/wiki/David_E._Goldberg (diakses pada 23 September 2012)
- http://www.davidgoldberg.com (diakses pada 23 September 2012)

c. Kenneth A. De Jong

Salah seorang murid John Holland yang berandil besar dalam pengembangan ilmu algoritma genetika adalah Kenneth A. De Jong, yang lebih dikenal dengan panggilan De Jong. Disertasinya, An Analysis of the Behaviour of a Class of genetic Adaptive Systems (1975), telah menjadi "master-piece" karena isinya mengombinasikan teori Schemata yang dicetuskan Holland dengan serangkaian hasil eksperimen komputer yang sangat teliti. Tahun 1984, De Jong bergabung dengan George Mason University dan menjabat sebagai profesor bidang ilmu komputer.

sumber:

- http://cs.gmu.edu/~kdejong/ (diakses pada 23 September 2012)
- http://cs.gmu.edu/~eclab/ (diakses pada 23 September 2012)
- http ://krasnow.gmu.edu/staff/ (diakses pada 23 September 2012)

d. Stephanie Forrest

Stephanie Forrest adalah wanita pertama yang ikut andil meramaikan perkembangan algoritma genetika. Ia adalah salah seorang murid John Holland yang mendapat bimbingan penuh darinya. Ia menyelesaikan PhD bidang Computer Scince di University of Michigan tahun 1985, tiga tahun setelah Goldberg meraih PhD-nya.e. Melanie Mitchell

Berbagai publikasi telah ditorehkan oleh Forrest, antara lain tentang tentang Immune System Modeling, Computer Security, Ecology and Artificial Life, Genetic Algoritms and Classifier System. Salah satu

penghargaan yang pernah diterima oleh Forrest adalah UNM Regents Lectureship dari University of New Mexico.

Sumber :

-http://www.santafe.edu/media/staff_cvs/ (diakses pada 23 September 2012)

-http://www.santafe.edu/about/people/ (diakses pada 23 September 2012)

-http://www.cs.unm.edu/~forrest (diakses pada 23 September 2012)

e. Melanie Mitchell

Melanie Mitchell adalah salah seorang murid bimbingan John Holland, selain Stephanie Forrest, yang juga memiliki andil dalam perkembangan ilmu algoritma genetika. Ia meraih gelar PhD bidang Computer Science dari University of Michigan pada tahun 1990 (lima tahun setelah Stepahine Forrest meraih gelar PhD-nya).

Berbagai pengalaman telah ditorehkan oleh Mitchell. Tahun 1990, ia menjadi asisten Profesor di departemen Electrical Engineering and Computer Science University of Michigan. ahun 1992–1999, dia menjabat sebagai *Research Profesor* di Santa Fe Institute. Tahun 1999–2000, beliau bergabung dengan Los Alamos National Laboratory. Tahun 2000–2002, beliau bergabung kembali dengan Santa Fe Institute. Setelah itu, di tahun 2002 juga, ia bergabung dengan Oregon Health & Science University, pada departemen Computer Science and Engineering.

Beberapa buku telah ditulis oleh Mitchell. Salah satu yang cukup terkenal adalah An Introduction to Genetic Algorithms (MIT Press 1996). Sementara yang lainnya adalah Analogy-Making as Perception (MIT Pres, 1993) and Adaptive Individuals in Evolving Populations : Models and Algorithms (Addison-Wesley 1996).

Sumber:

- -http://web.cecs.pdx.edu (diakses pada 23 September 2012)
- -http://web.cecs.pdx.edu/~mm (diakses pada 23 September 2012)
- -http://en.wikipedia.org/wiki/Melanie_Mitchell (diakses pada 23 September 2012)

f. Zbigniew Michalewicz

Zbigniew Michalewicz, 56 tahun, adalah salah seorang tokoh algoritma genetika kelahiran Polandia. Ia memperoleh gelar M.Sc. bidang Applied Mathematics (dari Technical University of Warsaw, Polandia) dan gelar PhD bidang Computer Science (dari Polish Academy of Science). Michalewicz telah menulis hampir 15 buku dan 200 artikel. Buku pertamanya yang sangat terkenal *Genetic Algorithm + Data Structures = Evolution Programs*, telah mendapat sambutan hangat dari kalangan akademisi dan praktisi.

Michalewicz memiliki pengalaman akademik dan non-akademik. Di bidang akademik, ia pernah mengajar di Institute of Computer Science - Polish Academy of Sciences (Polandia) dan University of North Carolina (Amerika). Selain itu, ia juga aktif menjadi editor senior di beberapa jurnal internasional. Baru-baru ini, ia ditunjuk sebagai *Executive Vice President* di IEEE Neural Network Council.

Di bidang non-akademik, ia pernah menjadi konsultan beberapa perusahaan ternama seperti IBM, Ford Motor Company, Teledyne Corporation. Kini, ia menjadi penasehat senior di NuTech Solutions, sebuah perusahaan jasa business solutions yang didirikannya tahun 1999 bersama seorang anaknya, Matthew Michalewicz. Di perusahaan ini, Michalewicz juga aktif melakukan penelitian dan pengembangan algoritma genetika, *fuzzy logic, neural network* untuk diaplikasikan pada

industri manufaktur dan jasa. Lawrence Davis, salah seorang rekan Michalewicz, juga aktif di perusahaan ini.

Sumber:

- http://cs.adelaide.edu.au/staff (diakses pada 23 September 2012)
- http://en.wikipedia.org/wiki/Zbigniew_Michalewicz (diakses pada 23 September 2012)
- http://www.solveitsoftware.com (diakses pada 23 September 2012)
- http://www.nutechsolutions.com (diakses pada 23 September 2012)

g. Kalyanmoy Deb

Kalyanmoy Deb lahir di Tripura, salah satu kota kecil di India. Masa kecilnya habiskan Calcutta. Tahun 1987, Deb mendapat kesempatan belajar S2 di luar negeri, di University of Alabama, Amerika. Di tempat inilah, Deb ditawari David Goldberg untuk ikut kursus algoritma genetikanya (kebetulan saat itu ia telah menyelesaikan dua semester di University of Alabama dan tengah mencari kursus di semester ketiga).

Merasa beruntung, Deb menerima tawaran tersebut dan kemudian melanjutkan studinya di University of Illinois at Urbana-Campaign (UIUC) untuk meraih gelar PhD. Setelah enam tahun tinggal di Amerika, tahun 1993 Deb kembali ke India dan bergabung dengan Indian Institute of Technology, Kanpur (IIT Kanpur) sebagai Assistant Professor. Saat tiba di tempat itu, ternyata IIT sudah menawarkan kursus tentang algoritma genetika dan cukup banyak juga mahasiswa yang tertarik mengambilnya. Akhirnya, pada tahun 1997, dengan bantuan kolega-koleganya di IIT, ia mendirikan pusat penelitian dan pengembangan ilmu algoritma genetika yang diberi nama Kanpur Genetic Algorithms Laboratory (KanGAL). Di laboratorium ini, telah

banyak dilakukan penelitian tentang aplikasi algoritma genetika, *fuzzy logic*, dan *neural networks* pada industri nyata.

Tahun 1991, setelah menyelesaikan draft program doktornya di UIUC, Deb menikah dengan Debjani, seorang wanita India yang juga tengah belajar di UIUC. Tahun 1994, mereka dikarunia seorang anak laki-laki dan kemudian tahun 1999 dikaruniai seorang anak perempuan .

Sumber:

http://www.iitk.ac.in (diakses pada 15 September 2012)

h. Mitsuo Gen

Mitsuo Gen adalah salah seorang tokoh algoritma genetika di Asia yang sudah mendunia. Beberapa karyanya, baik hasil penelitian maupun paper-paper, telah diterbitkan di beberapa jurnal internasional. Sementara itu, buku-buku yang ditulisnya pun mendapat sambutan yang hangat dari berbagai kalangan. Satu karyanya yang telah menjadi "pegangan wajib" bagi mahasiswa adalah Genetic Algorithms and Engineering Design, diterbitkan tahun 1997 oleh John Wiley & Sons.

Prof. Gen kini mengajar di Graduate School of Information, Productions, and Systems, Waseda University, Jepang. Di sana, ia memimpin Gen Lab, suatu lab *soft computing* yang mengkhususkan diri pada pengembangan komputasi evolusioner. Di Gen Lab ini terdapat juga beberapa *visiting profesor* seperti Dr. Fulya Altiparmak (Gazi University, Turki), Dr. YoungSu Yun (Daegu University, Korea), dan Dr. Shengxiang Yang (University of Leicester, UK).

Pengalaman mengajar Prof. Gen cukup banyak. Ia pernah menjadi dosen tamu di University of California, Texas A&M Univesity, University of Puerto Rico, dan University of Houston. Sementara itu, pengalaman menjadi editor jurnal internasional pun cukup bejibun. Ia pernah

menjadi editor di jurnal *Computational Intelligence of Computers and Industrial Engineering, Fuzzy Optimization and Decision Making, International Journal of Manufacturing Technology & Management*, dan *International Journal of Smart Engineering & Systems Design.*

Sumber:

-http://www.res.kutc.kansai-u.ac.jp (diakses pada 15 September 2012)

Daftar Pustaka

Apaiah RK, Hendrix EMT. 2004. Design of a supply chain network for pea-based novel protein foods, Journal of Food Engineering (available online at http://www.sciencedirect.com)

Arkeman Y, Dharma RA. 2008. Sistem Penunjang Keputusan Cerdas untuk Mengelola Rantai Pasokan pada Agroindustri Hortikultura. Institut Pertanian Bogor.

Arkeman Y. 1999. *Optimization of AGV-Based Flexible Manufacturing Systems Design Using Genetic Algorithms and Expert Systems.* PhD Thesis. Division of Information Technology, Engineering and The Environment. School of Engineering, Manufacturing and Mechanical Engineering, University of South Australia.

_____. 1996. *IFDES : An Intelligent System for Flexible Manufacturing Systems Design.* Master Thesis. School of Manufacturing and Mechanical Engineering, University of South Australia.

Arkeman Y, Luong L, Abhary K. *Design of Flexible Manufacturing Systems Using Genetic Algorithms and Artificial Intelligence.* International Journal of Flexible Automation and Integrated Manufacturing, 1999, pp. 145–171.

Arkeman Y, Anggraeni E. *Algoritma Genetika dan Penerapannya dalam Penyelesaian Masalah Tata Letak dan Desain Sistem Penanganan Bahan.* Jurnal Teknik Industri, Vol. 1 No. 1, Oktober 2000, pp. 1–8.

Arkeman Y, Prasetya H. *Algoritma Genetika dan Aplikasinya untuk Agribisnis Masa Depan.* AGRIMEDIA Volume 6 No. 1 Maret 2000.

Aryawan PT. 2003. *Penerapan Algoritma Tabu Search dalam Permasalahan Penjadwalan Job Shop.* Skripsi. Fakultas Teknik. Universitas Indonesia, Depok.

Askin RG, Charles R Standridge. 1993. *Modeling and Analysis of Manufacturing Systems.* John Wiley & Sons, Inc.

Austin JE. 1992. *Agroindustrial Project Analysis – Critical Design Factors.* Baltimore: The Johns Hopkins University Press.

Bagchi, Tapan P. 1999. *Multiobjective Scheduling By Genetic Algorithms.* United States of America: Kluwer Academic Publishers.

Bauer, Richard J. 1994. *Genetic Algorithms and Investment Strategies.* John Wiley & Sons, Inc.

Berry MJA, Gordon Linoff. 1997. *Data Mining Techniques for Marketing, Sales, and Customer Support.* John Wiley & Sons, Inc.

Black JT. 1991. *The Design of The Factory With a Future.* Singapore: McGraw-Hill International.

Bodenhofer U. 2003. Genetic Algorithms: Theory and Applications. Lectures Notes. www.flll.jku.at/div/teaching/Ga/GA-Notes.pdf (diakses pada 11 September 2012)

Bojadziev G, Bojadziev M. 1997. *Fuzzy Logic for Business, Finance, and Management.* World Scientific Publishing Co.

Brown JG, Deloitte, Touche. 1994. *Agroindustrial Investment and Operations.* The International bank for Recontruction and Development, USA.

Chen, KC Ian H, Cao A W. 2003. *A Genetic Algorithm for Minimum Tetrahedralization of a Convex Polyhedron.* CCCG 2003, Halifax, Nova Scotia. http://flame.cs.dal.ca:80/~cccg/papers/29/pdf. 20 Maret 2007.

Chong EKP, Zak SH. 1996. *An Introduction to Optimization.* John Wiley & Sons, Inc.

Chopra S, Meindl P. 2004. *Supply Chain Management: Strategy, Planning, and Operation.* United States of America: Pearson Prentice Hall, Inc.

Davis L. (Ed.). 1991. *Handbook of Genetic Algorithms.* New York: Van Nostrand Reinhold.

De Jong KA. 2006. Evolutionary Computation: A Unified Approach. Cambridge: MIT Press.

Dhar V, Stein R. 1997. *Intelligence Decision Support Methods: The Science of Knowledge Work.* United States of America: Pearson Prentice Hall, Inc.

Dietel HM, Dietel PJ, Steinbuhler K. 2001. *E-Business and e-Commerce for Managers.* New Jersey: Prentice Hall.

Evans JR, Olson DL. 2002. *Introduction to Simulation and Risk Analysis – second edition.* Upper Saddle River, New Jersey: Prentice-Hall.

Gen,M, Cheng R. 1997. *Genetic Algorithms and Engineering Design.* John Wiley & Sons, Inc.

Goldberg DE. 1989. *Genetic Algorithms in Search, Optimization, and Machine Learning.* Addison-Wesley, Reading, MA.

Heizer J, Render B. 2004. *Principles of Operations Management (fifth edition).* United States of America: Pearson Education, Inc.

Holland JH. *Genetic Algorithms.* Scientific American, Juli 1992.

Hopgood AA. 2001. *Intelligent Systems for Engineers and Scientists – second edition.* CRC Press LLC.

Hoover SV, Perry RF. 1989. *Simulation: A Problem Solving Approach.* Addison-Wesley, Reading, MA.

Jain N, Bagchi TP. *Hybridized GAs : Some New Results in Flowshop Scheduling Problems.* Indian Institute of Technology, Kanpur UP 208016, India.
http://www.citeseer.nj.nec.com
(diakses pada bulan Oktober 2002)

Karr CL, Freeman LM (Ed.). 1999. *Industrial Applications of Genetic Algorithms.* Boca Raton, Florida: CRC Press LLC.

Koza J. 2001. Genetic Algorithm. http://www.obitko.com/-tutorials/genetic-algorithms/introduction.php (diakses 11 September 2012)

Kusiak A. 1990. *Intelligent Manufacturing Systems.* Englewood Cliffs, NJ: Prentice-Hall

Kusumadewi S. 2003. Artificial Intteligence. Penerbit Graha Ilmu.

Levy, Simchi D, Kaminsky P, Levy Edith Simchi, 2000. *Designing and Managing the Supply Chain.* Singapore: Mc. Graw Hills Book Co.

Lin CT, Lee CSG. 1995. *Neural Fuzzy Systems : A Neuro – Fuzzy Synergism to Intelligent Systems.* Upper Saddle River, New Jersey: Prentice Hall.

Lotfi V, C Carl Pegels. 1996. *Decision Support Systems for Operations Management & Management Science – 3rd edition.* Richard D. Irwin, Inc.

Marimin. 2004. Teknik dan Aplikasi Pengambilan Keputusan Kriteria Majemuk. Jakarta: Grasindo.

McEllhiney RR (Ed.). 1994. *Feed Manufacturing Technology IV.* American Feed Industry Association, Inc.

Michalewicz Z. 1996. *Genetic Algorithms + Data Structures = Evolution Programs.* New York: Springer-Verlag.

Mitchell M. 2002. *An Introduction to Genetic Algorithms.* New Delhi: Prentice-Hall of India.

Nicholas JM. 1990. *Managing Business &Engineering Projects.* Prentice-Hall, Inc.

Petrovic S. *Automated Scheduling.* School of Computer Science and Information Technology, The University of Nottingham, UK. http://www.cs.nott.ac.uk/~sxp/scheduling (diakses pada bulan Oktober 2002)

Reeves CR, Yamada T. 1996. *Genetic Algorithms, Path Relinking and The Flowshop Sequencing Problem.* http://www.citeseer.nj.nec.com (diakses pada bulan Januari 2002)

Riswan. 1993. *Rancangan Program Komputer untuk Penjadwalan Job Shop dalam Perencanaan Paket Pesanan di PT DC dengan Metode Priority Dispatching.* Skripsi. Fakultas Teknik. Universitas Indonesia, Depok.

Satrio AB, Arkeman Y. 2007. Optimasi Masalah Penjadwalan *Job-Shop* untuk Industri Peralatan Pengolahan Hasil Pertanian dengan Menggunakan Algoritma Genetika. Institut Pertanian Bogor.

Setiawan S. 1993. *Artificial Intelligence.* Yogyakarta: ANDI Offset.

Sidharta F. 2005. Desain Algoritma Genetika untuk Optimasi Penjadwalan Produksi Meubel Kayu. Tesis. Program Masgiter Ilmu Komputer, FMIPA IPB.

Turban E, Aronson JE. 2001. *Decision Support Systems and Intelligent Systems – sixth edition.* Upper Saddle River, New Jersey: Prentice Hall.

Turban E, Meredith JR. 1991. *Fundamentals of Management Science-5th ed.* IRWIN Homewood IL, Boston.

Underwood L. 1994. *Intelligent Manufacturing.* Addison-Wesley Publishing Company, Inc.

Vose MD. 1999. *The Simple Genetic Algorithms: Foundation and Theory.* Massachusets Institute of Technology, USA.

Lampiran

Lampiran 1. *Source Code* Permainan Optimasi Fungsi Sulit

```
Program GA-1;

{Complex Function Optimization using GAs}
{Reference : Goldberg, 1989}
{Binary number is used for chromosome representation,
 one-point crossover is used for crossover, and swap method
is used for mutation.}

{This GA is used to optimize Fit (x)= x.sin(10#.x) + 11 ;  -
1<=x<=2 ;
 ChromLength = 22  -->   the precision value of f(x) is 10^-
6 }
{Re-programmed by : Yandra Arkeman & Hendra Gunawan  }


Uses WinCrt;
Const MaxPop = 100;
      MaxString = 30;

Type Allele = boolean;
     Chromosome = array [1..MaxString] of Allele;
     Individual = Record
                    Chrom : Chromosome;
                    Decimal, x, Fitness : real;
                    Parent1, Parent2, XSite : integer;
                    tanda : char;
                End;
     Population = array [1..MaxPop] of Individual;
Var
    OldPop, NewPop : Population;
    PopSize, LChrom, Gen, MaxGen : integer;
    PCross, PMutate : real;
    NMutation, NCross, JCross, JMutate : integer;
    SumFit, AvgFit, MaxFit, MinFit, Decim : real;
    OldRand : array [1..55] of real;     {array of 55 random
number}
    JRand : integer;              {Current random}
    ReportAll, ReportGen : text;
```

```
    RandomSeed : real;

Procedure Advance_Random;
{Create next batch of 55 random number}
   Var jl : integer;
       New_Random : real;
   Begin
      For jl:=1 to 24 do
      Begin
        New_Random := OldRand [jl] - OldRand[jl+31];
        If (New_Random < 0.0 ) then New_Random := New_Random
+ 1.0;
        OldRand [jl] := New_Random;
      End;

      For jl := 25 to 55 do
      Begin
        New_Random := OldRand[jl] - OldRand[jl-24];
        If (New_Random < 0.0 ) then New_Random := New_Random
+ 1.0;
        OldRand[jl] := New_Random;
      End;
   End;

Procedure WarmUp_Random (Random_Seed : real);
{get random off and runnin}
   Var jl, ii : integer;
       New_Random , Prev_Random : real;
   Begin
      OldRand[55] := Random_Seed;
      New_Random := 1.0E-9;
      Prev_Random := Random_Seed;

      For jl := 1 to 54 do
      Begin
        ii := 21*jl mod 55  ;
        OldRand [ii] := New_Random;
        New_Random := Prev_Random - New_Random;
        If (New_Random < 0.0 ) then New_Random := New_Random
+ 1.0;
        Prev_Random := OldRand[ii];
      End;
      Advance_Random; Advance_Random; Advance_Random;
      JRand := 0;
   End;

Function Random : real;
{fetch a single random number between 0.0 and 1.0 -
Substractive Methods}
   Begin
```



```
      JRand := JRand + 1;
      If (JRand > 55 ) then
      Begin
        JRand := 1;
        Advance_Random;
      End;
      Random := OldRand[JRand];
   End;


Function Flip (probability : real) : boolean;
{Flip a biased coin - true if heads}
   Begin
      If probability = 1 then Flip := True
      Else Flip := (Random <= probability);
   End;

Function Rnd (Low, High : integer) : integer;
{Pick a random integer between Low and High }
   Var i : integer;
   Begin
      if Low >= High then i := Low else
      Begin
        i := Trunc (Random * (High-Low+1) + Low);
        If i > High Then i := High;
      End;
      Rnd := i;
   End;
Procedure Randomize;
{Get seed number for random and start it up}
   {Var RandomSeed : real;}
   Begin
      Repeat
        Write ('Enter seed random number (0.0 ... 1.0) = '
); readln (RandomSeed);
      Until (RandomSeed > 0) and (RandomSeed < 1.0);
      WarmUp_Random (RandomSeed);
   End;


Function FitFunc (x : real ) : real;
  {fitness function f(x)=x* Sin(10#.x) + 11}
  Begin
     FitFunc := x * Sin (10* 3.14159265358979323846264 3* x)
+ 11;
       {1 phi = 3.14159265358979323846264 3  }
  End;
```

```
Function Decode (Chrom : Chromosome; LChrom : integer) :
real;
   {Decode string as unsigned binary integer, true=1,
false=0}
   Var j : integer;
       Accum, PowerOf2 : real;
   Begin
      Accum := 0.0;
      PowerOf2 := 1;
      For j:=1 to LChrom do
      Begin
        If Chrom[j] Then Accum := Accum + PowerOf2;
        PowerOf2 := PowerOf2 * 2;
      End;
      Decode := Accum;
   End;


Procedure Statistics (PopSize : integer ; var MaxFit,
AvgFit, MinFit, SumFit : real; Var Pop : Population);
   {Calculate population ststistics}
   Var j : integer;
   Begin
      {Initialize}
      SumFit := Pop[1].Fitness;
      MinFit := Pop[1].Fitness;
      MaxFit := Pop[1].Fitness;

      {Loop for Max, Min, & SumFitness for EACH GENERATION }
      For j:= 2 to PopSize do
        With Pop[j] do
        Begin
           SumFit  := SumFit + Fitness;
           If Fitness > MaxFit then MaxFit := Fitness;
           If Fitness < MinFit then MinFit := Fitness;
        End;

      {Calculate fitness average }
      AvgFit := SumFit / PopSize;
   End;
Procedure Initdata;
{Interactive data inquiry and set up }
   Begin
      ClrScr;
      Writeln;
      Writeln (' GENETIC ALGORITHMs for Complex Func.
Optimization');
      Writeln
('=================================================== ');
      Writeln;
```



```
      Writeln ('This GAs is used to optimize Fit(x)=
x.sin(10#.x) + 11 ; -1<=x<=2');
      Writeln ('Chrom. representation : Binary
Representation ');
      Writeln ('Crossover             : One-point Crossover
');
      Writeln ('Mutation              : Swap Mutation ');
      Writeln ('Selection             : Roulette-Wheel
Selection');
      Writeln;
      Writeln;
      Writeln ('DATA INPUT');
      Writeln ('==========');
      Repeat
       Write ('Enter Pop. Size (Max=100)    : '); readln
(PopSize);
      Until (PopSize > 0) and (PopSize < 100);
      {Repeat
       Write ('Enter Chrom. Length (Max=30) : '); readln
(LChrom);
      Until (LChrom > 0) and (LChrom < 30);   }

      LChrom:=22;
             {---> the chrom length can be changed,
             depend on the precision value needed}
      Repeat
       Write ('Enter Max. Generation       : '); readln
(MaxGen);
      Until MaxGen > 0;
      Repeat
       Write ('Enter Crossover probability (0 - 1): ');
readln (PCross);
      Until (PCross >= 0) and (PCross <= 1);
      Repeat
       Write ('Enter Mutation probability (0 - 1): ');
readln (PMutate);
      Until (PMutate >=0) and (PMutate <= 1);
      Randomize;
      NMutation := 0;
      NCross := 0;
   End;


Procedure InitReport ; {Initial Report}
   Const chroms = 'Chromosome';
        Decim = 'Decimal';
        x = 'x';
        Fit = 'Fitness';
   Var j, jl : integer;  ReportInit : text;
   Begin
```

```
      Assign (ReportInit,'C:\Bp\kgun\GA1\File_1.txt');
      Rewrite (ReportInit);
      Writeln
(ReportInit,'=====================================');
      Writeln (ReportInit,'            INITIAL REPORT
');
      Writeln (ReportInit,' Simple Function Optimization
using GAs');
      Writeln (ReportInit,'  Fit(x)= x.sin (10#.x) + 11 ; -
1<=x<=2');
      Writeln
(ReportInit,'=====================================');
      Writeln (ReportInit);
      Writeln (ReportInit,'GAs parameters');
      Writeln (ReportInit,'==============');
      Writeln (ReportInit,'Pop. Size           = ',PopSize);
      Writeln (ReportInit,'Chrom. Length       = ',LChrom);
      Writeln (ReportInit,'Max. Generation     = ', MaxGen);
      Writeln (ReportInit,'Crossover probability   =
',PCross:4:4);
      Writeln (ReportInit,'Mutation probability    =
',PMutate:4:4);
      Writeln (ReportInit,'Seed Random Number      =
',RandomSeed:4:4);
      Writeln (ReportInit);
      Writeln (ReportInit);
      Writeln (ReportInit,'INITIAL GENERATION STASTISTICS');
      Writeln
(ReportInit,'============================');
      Writeln (ReportInit,' ':18-Length(chroms), '
',chroms,'  ':10,Decim,'   ':10,x,' ':8,Fit);

      For j:= 1 to PopSize do

      With OldPop[j] do
        Begin
          Write (ReportInit,j:3,') ');
          For Jl := LChrom downto 1 do
            Begin
            If Chrom [jl] Then Write (ReportInit,'1')
            Else Write (ReportInit,'0');
            End;
          Writeln (ReportInit,'    ',Decimal:10:0,'
',x:10:6,'   ',Fitness:10:6);
        End;
      Writeln (ReportInit);
      Writeln (ReportInit,'Sum of Fitness = ',SumFit:10:6);
      Writeln (ReportInit,'Max. Fitness   = ',MaxFit:10:6);
      Writeln (ReportInit,'Min. Fitness   = ',MinFit:10:6);
      Writeln (ReportInit,'Avg. Fitness   = ',AvgFit:10:6);
```



```
      Writeln (ReportInit);
      Writeln (ReportInit);
      Close (ReportInit);
   End;

Procedure InitPop;
   Var j, jl : integer;
   Begin
      For j:=1 to PopSize do
      With OldPop[j] do
      Begin
        For jl:= 1 To LChrom do
        Chrom[jl] := Flip(0.5);      {a fair coin toss}
        Decimal := Decode (Chrom,LChrom); {Decode the
string}
        x := -1 + (Decimal/4194303)*3;    {corresponding
value for variable x}
        Fitness := FitFunc(x);
        Parent1 := 0;
        Parent2 := 0;
        XSite   := 0;
      End;
   End;


Procedure Initialize;   Begin
      Initdata;
      InitPop;
      Statistics (PopSize, MaxFit, AvgFit, MinFit, SumFit,
OldPop);
      InitReport;
   End;

Function Select (PopSize : integer; SumFit : real; Var Pop :
Population) : integer;
   {Select a single individual via roulette wheel selection
}
   Var Rand, PartSum : real;  {random point an a wheel,
partial sum}
       j : integer;
   Begin
      PartSum := 0.0;  {Zero outcounter & accumulator}
      j:= 0;
      Rand := Random * SumFit; {Wheel point calc uses random
number  [0,1] }
      Repeat  {Find wheel slot}
        j := j+1;
        PartSum := PartSum + Pop[j].Fitness;
      Until (PartSum >= Rand ) or (j=PopSize);
```

```
      {Return individual number}
      Select := j;
   End;

Function Mutation (AllelEval : Allele; PMutate : real; Var
NMutation : integer ) : Allele;
   Var Mutate : boolean;
   Begin
      Mutate := Flip (PMutate);  {Flip the biased coin}
      If Mutate Then
      Begin
         NMutation := NMutation + 1;
         Mutation := Not AllelEval {Change bit value, True
become false, False become True}
      End
      Else
      Mutation := AllelEval;
   End;


Procedure Crossover (var p1, p2, Child1, Child2 :
Chromosome;
                     Var LChrom, NCross, JCross : integer;
                     Var PCross : real);
   {Cross 2 parents with one-point crossover methods}
   Var j,k : integer;
   Begin
      JCross := 0;
      If Flip (PCross) then
      Begin
        JCross := Rnd(1, LChrom-1);
        NCross := NCross + 1;

         JCross := LChrom-JCross; { Xsite from left }
      End;

         For j:= LChrom downto JCross do
         Begin
           Child1[j] := p1[j];
           Child2[j] := p2[j];
         End;

         For k:= JCross+1 downto 1 do
       Begin
          Child1[k] := p2[k];
          Child2[k] := p1[k];
       End;
   End;

Procedure Generation;
```



```
   {Create a new generation through Select, Crossover, and
Mutation }
   {Note : generation assumes an even-numbered PopSize
   Var j, Mate1, Mate2, JCross : integer;
       td : char; RChar : integer;
   Begin
      j := 1;
      Repeat   {Select, Crossover, and Mutation until NewPop
is filled}
        Mate1 := Select (PopSize, SumFit, OldPop);
{Pick a pair of mates}
        Mate2 := Select (PopSize, SumFit, OldPop);

        {Crossover and Mutation  --- Mutation embedded
within Crossover}
        Crossover (OldPop[Mate1].Chrom, OldPop[Mate2].Chrom,
                   NewPop[j].Chrom, NewPop[j+1].Chrom,
                   LChrom, NCross, JCross, PCross);

         {Decode string, evaluate fitness, & record
parentage date on both children}
         td :=' ';
        With NewPop[j] do
        Begin
            If Flip (PMutate) Then
            Begin
               JMutate := Rnd(1, LChrom);
              NMutation := NMutation+1;

               Chrom[JMutate]:= Not Chrom[JMutate];
               td := '$';
              tanda := td;
           End;

           Decimal := Decode (Chrom, LChrom);
           x := -1 + (Decimal/4194303)*3;    {corresponding
value for variable x}
           Fitness := FitFunc(x);
           Parent1 := Mate1;
           Parent2 := Mate2;
           XSite   := JCross;
           tanda   := td;
        End;
        td := ' ';

        With NewPop[j+1] do
        Begin
           If Flip (PMutate) Then
            Begin
               JMutate := Rnd(1,LChrom);
```

```
              NMutation := NMutation+1;

              Chrom[JMutate]:= Not Chrom[JMutate];
              td := '$';
              tanda := td;
            End;

            Decimal := Decode (Chrom, LChrom);
           x := -1 + (Decimal/4194303)*3;     {corresponding
value for variable x}
           Fitness := FitFunc(x);
           Parent1 := Mate1;
           Parent2 := Mate2;
           XSite   := JCross;
            tanda   := td;
       End;
        td := ' ';
       {Increment population index }
       j := j + 2;     {jml populasi GENAP}
     Until j>PopSize;
  End;


Procedure Report (Gen : integer) ;
{Write the population report}
  Const Chrome = '< Chrom. >';
        Dec    = '<Decimal>';
        x_val  = '< x >';
        Fitn   = '<Fitness>';
      p1p2   = '<p1,p2>';
        XSit   = '<XSite>';
        Ch1Ch2 = '< Ch1&Ch2 >';

  Var j, jl : integer;
  Begin
     Append (ReportAll);
     Writeln (ReportAll,'Generation ',gen-1:3, ' and
generation ', gen:3);
     Write (ReportAll,' ':18-Length(Chrome),Chrome, '
':11,Dec, ' ':5,x_val, ' ':3,Fitn, ' ':10,p1p2);
     Writeln (ReportAll,' ':2,XSit, ' ':17-
Length(Ch1Ch2),Ch1Ch2, ' ':11,Dec,' ':5,x_val, ' ':5,Fitn);
     For j:= 1 to PopSize do
     Begin
       Write (ReportAll,j:3,') ');

       With OldPop [j] do
       Begin
          For jl:= LChrom downto 1 do
           Begin
```



```
            If Chrom [jl] Then Write (ReportAll,'1')
            Else Write (ReportAll,'0');
          End;
         Write (ReportAll,' ',Decimal:10:0,' ',x:10:6,'
',Fitness:10:6, '  ',tanda);
       End;

       With NewPop[j] do
       Begin
          Write (ReportAll,'  ',j:2,')   (',Parent1:2, ',',
Parent2:2, ')','    ',XSite:2, '      ');

          For jl:= LChrom downto 1 do
              Begin
             If Chrom[jl]Then Write (ReportAll,'1')
             Else Write (ReportAll,'0');
              End;

         Writeln (ReportAll,'  ',Decimal:10:0,'  ',x:10:6,
'  ',Fitness:10:6, ' ', tanda);
       End;
     End;
     Writeln (ReportAll);
     Writeln (ReportAll,'SumFitness Value of generation
',gen:4,' = ',SumFit:10:6);
     Writeln (ReportAll,'Max. Fitness Value of generation
',gen:4,' = ',MaxFit:10:6);
     Writeln (ReportAll,'Min. Fitness Value of generation
',gen:4,' = ',MinFit:10:6);
     Writeln (ReportAll,'Avg. Fitness Value of generation
',gen:4,' = ',AvgFit:10:6);
     Writeln (ReportAll,'Cum. of NCross until gen. ',gen,' =
',NCross);
     Writeln (ReportAll,'Cum. of NMutation until gen.
',gen,' = ',NMutation);
     Writeln (ReportAll);
  End;


Procedure ReportGeneration (gen:integer) ;
{write the generation report}
   Var j : integer;
   Begin
      Append (ReportGen);
      Begin
         Write (ReportGen, gen:4,') ',SumFit:10:6,'   ');
         Write (ReportGen,MaxFit:10:6,'   ');
         Write (ReportGen,MinFit:10:6,'   ');
         Write (ReportGen,AvgFit:10:6,'    ');
         {Write (ReportGen,Decim:10:6,'     ');}
```

```
            Write (ReportGen,NCross:5,'    ');
            Writeln (ReportGen,NMutation:5);
         End;
      End;



BEGIN    {PROGRAM UTAMA}

   Clrscr;
   Assign (ReportAll,'C:\Bp\kgun\GA1\File_2.txt');
   Rewrite (ReportAll);
   Assign (ReportGen,'C:\Bp\kgun\GA1\File_3.txt');
   Rewrite (ReportGen);
   Gen := 0;
   Initialize;
   Writeln ; Writeln;
   Writeln ('Wait for a moment, please...  );
   Writeln;
   Writeln ('Writing 3 output files in C:\Bp\kgun\GA1\...');
   Writeln ('    File_1 : Initial report');
   Writeln ('    File_2 : Detail report');
   Writeln ('    File_3 : Summary report');
   Repeat
      Gen := Gen + 1;
      Generation;
      Statistics (PopSize, MaxFit, AvgFit, MinFit, SumFit,
NewPop);
      Report (Gen);
      ReportGeneration (Gen);
      OldPop := NewPop;  {Advance the generation}
   Until (Gen >= MaxGen);
   Close (ReportAll);
   Close (ReportGen);
   Writeln;  Writeln;
   Writeln ('FINISHED.');
   Writeln ('The output files in C:\Bp\kgun\GA1\...');

END.
```


Profil Penulis

Dr. Ir. Yandra Arkeman, MEng

Lahir di Payakumbuh 14 September 1965. Lulusan Jurusan Teknologi Industri Pertanian, Institut Pertanian Bogor (IPB) tahun 1989 dengan skripsi tentang *Computer Aided Quality* (CAQ) untuk Industri Minuman Ringan. Tahun 1994 mulai melanjutkan studi untuk jenjang S-2/S-3 di Adelaide, Australia. Meraih gelar Master (1996) dan Doktor (2000) dengan riset di bidang *Intelligent Manufacturing Systems* dari University of South Australia. Selanjutnya pada tahun 2004–2006 melakukan penelitian tingkat *post-doctoral* di Applied System Design Laboratory, Department of Electrical Engineering and Computer Science, Kansai University, Osaka, Japan dengan topik *Multi-objective Genetic Algorithms for Agroindustrial Supply Chain Design*. Kemudian pada tahun 2009 mendapat kesempatan untuk melakukan penelitian *post-doctoral* yang kedua selama 4 bulan di Department of Computer Science dan Krasnow Institute for Advanced Study, George Mason University, Virginia, USA dengan topik tentang *Parallel Genetic Algorithms for Agroindustrial System Design*. Pada tahun 2003 mendirikan CIGARIS (Computational Intelligence Group for Advanced Research and Innovations in Supercomputing Technology) dengan alamat web: www.cigaris.com. Sejak tahun 2007 menjadi peneliti pada Surfactan and Bioenergy Research Centre (SBRC), IPB, dengan fokus penelitian pada Intelligent Modeling for Bioenergy Sustainability.

Aktif di berbagai organisasi profesi di tingkat nasional dan internasional sejak tahun 1994, di antaranya, Institute of Industrial Engineers (IIE), Society of Manufacturing Engineers (SME) dan International Society of Systems Science (ISSS). Riwayat hidupnya dimasukkan di buku biografi *Who's Who in Science and Engineering 1998/1998* terbitan Marquis Publication, USA.

Menikah dengan Ir. Titi Sulistyowati Rahayu dan dikarunia satu orang putra yang bernama Ryan Muhammad Khawarizmi, mahasiswa Teknik Fisika ITB tahun 2011. Selain melakukan penelitian, waktu luangnya diisi dengan bermain tenis dan *traveling*.

Prof. Dr. Ir. Kudang Boro Seminar, M.Sc

Lahir di Jember 18 Nopember 1959. Guru Besar bidang Teknologi Komputer di Departemen Teknik Pertanian, Fakultas Teknologi Pertanian (FATETA) dan Departemen Ilmu Komputer, Fakultas Matematika dan Ilmu Pengetahuan Alam (FAMIPA), IPB. Menyelesaikan studi S-1 di IPB tahun 1983, dan S-2 serta S-3 di Faculty of Computer Science University of New Brunswick Canada pada tahun 1989 dan 1993. Bidang riset yang ditekuni mencakup *Information Engineering, Software Engineering, Intelligent Systems, Distance Learning, Internetworking, Computer-Based Ins-trumentation & Control Systems*. Sejak menyelesaikan studi doktornya, mendapat amanah untuk menjadi Ketua Departemen Teknik Pertanian IPB (1997–2000), Ketua Program Studi Pasca Sarjana Ilmu Keteknikan Pertanian IPB (2000–2003), Kepala Bagian (Lab) Ergotron(2008-kini), Kepala Perpustakaan IPB (2003–2007), dan Direktur Komunikasi dan Sistem Informasi IPB (2007–2012). Terlibat dalam tim desain & implementasi pembentukan

Departemen Ilmu Komputer IPB, Program Studi Magister Komputer IPB, Program Studi Manajemen Teknologi Informasi untuk Perpustakaan IPB, pembukaan program Doktoral Jalur Riset Ilmu Keteknikan Pertanian IPB, serta pembentukan rumpun Departemen Teknik di IPB. Dalam bidang keprofesian adalah Ketua **HIPI/ISAI (Himpunan Informatikan Pertanian Indonesia/***Indonesian Society of Agriculture Informatics***)** , *honorary member of* **AFITA** (*Asian Federation for Information Technology in Agriculture*), dan anggota PERTETA (**Perhimpunan Teknik Pertanian/***Indonesian Society of Agricultural Engineering*). Kesempatan menggali ilmu yang sangat berharga adalah kesempatan menimba dan mendalami ilmu agama khususnya Al-Qur'an baik dalam membaca dan mengkajinya sejak tahun 1996 hingga saat ini. Melalui bimbingan guru-guru yang yang bersahaja (*tawadhu'*) dalam ketinggian ilmunya yang salah satunya berperingkat hafiz (penghafal Al-Qur'an) tanpa meninggalkan profesi sebagai akademisi, peneliti, dan pendidik.

Hendra Gunawan, S.TP

Lahir di Jakarta, 12 April 1980. Lulusan dari Departemen Teknologi Industri Pertanian, Fakultas Teknologi Pertanian, Institur Pertanian Bogor, pada tahun 2003. Ia menyelesaikan gelar kesarjanaannya dengan melakukan riset tentang aplikasi Algoritma Genetika untuk penjadwalan *flow-shop* bidang agroindustri. Bidang area yang menjadi ketertarikannya antara lain Teknologi Informasi dan Komunikasi (TIK), Sistem Intelijen, Kecerdasan Buatan, dan Manajemen Perbankan.

Beberapa kursus pernah diikutinya antara lain Kursus Akuntansi Terapan (STAN Bintaro), Kursus Perpajakan Brevet A-B (STAN Bintaro), dan kursus CCNA-Cisco Certified Network Associates (Junicorp-

Trisakti). Sejak tahun 2005, ia bekerja di PT Bank Rakyat Indonesia (Persero), Tbk di Kantor Cabang Tangerang. Beberapa artikel yang ditulisnya pernah dimuat di tabloid PCplus dan majalah TSI (Teknologi dan Sistem Informasi)-BRI. Saat ini, ia sedang melanjutkan studi pascasarjana di Universitas Indonesia, jurusan Magister Teknologi Informasi.