BAB I
KECERDASAN BUATAN

A. Pendahuluan

Di masa kini, Kecerdasan Buatan (*Artificial Intelligence*, AI) telah menjadi wacana umum yang sangat penting dan jamak dijumpai. Namun masih banyak menyisakan pertanyaan skeptis tentang ‘mesin berfikir’: “*Betulkah sebuah mesin dapat benar-benar berfikir dengan dirinya sendiri?*”, atau “*Jika benar-benar dapat berfikir sendiri, apakah proses berfikirnya sama dengan kita?*”, dan “*Seberapa handal?*”.

1. Sejarah Kecerdasan Buatan

Di awal abad 20, seorang penemu Spanyol, Torres y Quevedo, membuat sebuah mesin yang dapat men’skak-mat’ raja lawannya dengan sebuah ratu dan raja. Perkembangan secara sistematis kemudian dimulai segera setelah diketemukannya komputer digital. Artikel ilmiah pertama tentang Kecerdasan Buatan ditulis oleh Alan Turing pada tahun 1950, dan kelompok riset pertama dibentuk tahun 1954 di Carnegie Mellon University oleh Allen Newell and Herbert Simon. Namun bidang Kecerdasan Buatan baru dianggap sebagai bidang tersendiri di konferensi Dartmouth tahun 1956, di mana 10 peneliti muda memimpikan mempergunakan komputer untuk memodelkan bagaimana cara berfikir manusia. Hipotesis mereka adalah: “*Mekanisme berfikir manusia dapat secara tepat dimodelkan dan disimulasikan pada komputer digital*”, dan ini yang menjadi landasan dasar Kecerdasan Buatan.

2. Definisi Kecerdasan Buatan

Tidak ada kesepakatan mengenai definisi Kecerdasan Buatan, di antaranya adalah:

a. Sebuah studi tentang bagaimana membuat komputer mengerjakan sesuatu yang dapat dikerjakan manusia (Rich, 1991)
b. Cabang ilmu komputer yang mempelajari otomatisasi tingkah laku cerdas (Setiawan, 1993)
c. Suatu perilaku sebuah mesin yang jika dikerjakan oleh manusia akan disebut cerdas (Turing, et. al, 1996)

Kebanyakan ahli setuju bahwa Kecerdasan Buatan berhubungan dengan 2 ide dasar. Pertama, menyangkut studi proses berfikir manusia, dan kedua, berhubungan dengan merepresentasikan proses tersebut melalui mesin (komputer, robot, dll)

Kemampuan untuk *problem solving* adalah salah satu cara untuk mengukur kecerdasan dalam berbagai konteks. Terlihat di sini bahwa mesin cerdas akan diragukan untuk dapat melayani keperluan khusus jika tidak mampu menangani permasalahan remeh/kecil yang biasa dikerjakan orang secara rutin. Terdapat beberapa alasan untuk memodelkan performa manusia dalam hal ini:

a. Untuk menguji teori psikologis dari performa manusia
b. Untuk membuat komputer dapat memahami penalaran (*reasoning*) manusia
c. Untuk membuat manusia dapat memahami penalaran komputer
d. Untuk mengeksploitasi pengetahuan apa yang dapat diambil dari manusia

Menurut Winston dan Prendergast (1984), tujuan dari Kecerdasan Buatan adalah:

a. Membuat mesin menjadi lebih pintar.
b. Memahami apakah kecerdasan (*intelligence*) itu.
c. Membuat mesin menjadi lebih berguna.

3. Kecerdasan

Dari kamus, arti kecerdasan adalah: kemampuan untuk mengerti/memahami (*The faculty of understanding*). Perilaku cerdas dapat ditandai dengan:

a. Belajar atau mengerti dari pengalaman
b. Memecahkan hal yang bersifat mendua atau kontradiktif
c. Merespon situasi baru dengan cepat (fleksibel)
d. Menggunakan alasan untuk memecahkan problem secara efektif
e. Berurusan dengan situasi yang membingungkan
f. Memahami dengan cara biasa/rasional
g. Menerapkan pengetahuan untuk memanipulasi lingkungan
h. Mengenali elemen penting pada suatu situasi

Sebuah ujian yang dapat dilakukan untuk menentukan apakah sebuah komputer/ mesin menunjukkan perilaku cerdas didesain oleh Alan Turing. Tes Turing menyatakan sebuah mesin dikatakan pintar hanya apabila seorang pewawancara (manusia) yang berbicara dengan orang lain dan mesin yang dua-duanya tidak terlihat olehnya, tidak

mampu menentukan mana yang manusia dan mana yang mesin, meskipun dia telah berulang-ulang melontarkan pertanyaan yang sama.

4. Perbandingan AI dengan program komputer konvensional

Program komputer konvensional prosesnya berbasis algoritma, yakni formula matematis atau prosedur sekuensial yang mengarah kepada suatu solusi. Algoritma tersebut dikonversi ke program komputer yang memberitahu komputer secara pasti instruksi apa yang harus dikerjakan. Algoritma yang dipakai kemudian menggunakan data seperti angka, huruf, atau kata untuk menyelesaikan masalah.

Perangkat lunak AI berbasis representasi serta manipulasi simbolik. Di sini simbol tersebut berupa huruf, kata, atau angka yang merepresentasikan obyek, proses dan hubungan keduanya. Sebuah obyek bisa jadi seorang manusia, benda, pikiran, konsep, kejadian, atau pernyataan suatu fakta. Menggunakan simbol, kita dapat menciptakan basis pengetahuan yang berisi fakta, konsep, dan hubungan di antara keduanya. Kemudian beberapa proses dapat digunakan untuk memanipulasi simbol tersebut untuk menghasilkan nasehat atau rekomendasi untuk penyelesaian suatu masalah. Perbedaan dasar antara AI dengan program komputer konvensional diberikan dalam Tabel I-1.

Tabel I-1. Perbandingan antara AI dan Program Konvensional

Aspek	AI	Program konvensional
Pemrosesan	Sebagian besar simbolik	Algoritmik
Input	Tidak harus lengkap	Harus lengkap
Pendekatan pencarian	Sebagian besar heuristik	Algoritma
Penjelasan/eksplanasi	Tersedia	Biasanya tidak tersedia
Fokus	Pengetahuan	Data
Pemeliharaan & peningkatan	Relatif mudah	Biasanya sulit
Kemampuan berpikir	Ada	Tidak ada

secara logis		

B. Cabang Kecerdasan Buatan

Pencarian. Program AI seringkali harus mengevaluasi kemungkinan yang jumlahnya banyak sekali, misalnya kemungkinan langkah dalam permainan catur atau penyimpulan dari program untuk membuktikan suatu teori.

Pengenalan Pola.

Representasi, yakni bagaimana merepresentasikan/menuliskan fakta-fakta yang ada ke dalam simbul-simbul atau bahasa logika matematis.

Inferensi.

Pengetahuan dan penalaran yang masuk akal (*common sense knowledge and reasoning*).

Belajar dari pengalaman.

Perencanaan. Program perencanaan bermula dari fakta-fakta umum (terutama fakta mengenai efek dari suatu aksi), fakta tentang situasi yang khusus, dan suatu pernyataan tentang tujuan. Dari sini kemudian dibuat sebuah strategi untuk mencapai tujuan tersebut. Secara umum, biasanya strategi tersebut berupa urut-urutan aksi.

Epistemologi, yakni studi tentang sumber, sifat, dan keterbatasan pengetahuan yang digunakan untuk pemecahan masalah.

Ontologi, ilmu tentang keberadaan dan realitas.

Heuristik, yaitu suatu cara atau teknik untuk mencoba menemukan suatu benda/ide.

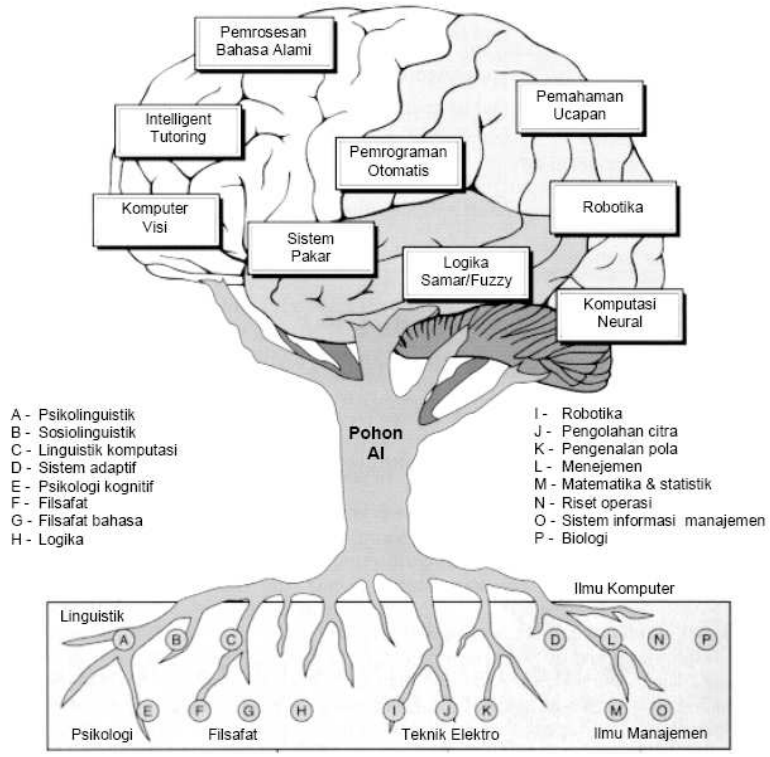
C. Bidang Aplikasi Kecerdasan Buatan

Penerapan Kecerdasan Buatan meliputi berbagai bidang seperti ditunjukkan pada bagian akar pohon AI dalam Gambar I-1, antara lain: Bahasa/linguistik, Psikologi, Filsafat, Teknik Elektro, Ilmu Komputer, dan Ilmu Manajemen. Sedangkan sistem cerdas yang banyak dikembangkan saat ini adalah:

Sistem Pakar (*Expert Systemi),* yaitu program konsultasi (*advisory*) yang mencoba menirukan proses penalaran seorang pakar/ahli dalam memecahkan masalah yang rumit. Sistem Pakar merupakan aplikasi AI yang paling banyak. Lebih detil tentang Sistem Pakar akan diberikan dalam bab berikutnya.

Pemrosesan Bahasa Alami (*Natural Language Processing*), yang memberi kemampuan pengguna komputer untuk berkomunikasi dengan komputer dalam bahasa mereka sendiri (bahasa manusia). Sehingga komunikasi dapat dilakukan dengan cara percakapan alih-alih menggunakan perintah yang biasa digunakan dalam bahasa komputer biasa. Bidang ini dibagi 2 lagi:

a. Pemahaman bahasa alami, yang mempelajari metode yang memungkinkan komputer mengerti perintah yang diberikan dalam bahasa manusia biasa. Dengan kata lain, komputer dapat memahami manusia.
b. Pembangkitan bahasa alami, sering disebut juga sintesa suara, yang membuat komputer dapat membangkitkan bahasa manusia biasa sehingga manusia dapat memahami komputer secara mudah.



Gambar I-1. Pohon Kecerdasan Buatan dan aplikasi utamanya

Pemahaman Ucapan/Suara (*Speech/Voice Understanding*), adalah teknik agar komputer dapat mengenali dan memahami bahasa ucapan. Proses ini mengijinkan seseorang berkomunikasi dengan komputer dengan cara berbicara kepadanya. Istilah “pengenalan suara” mengandung arti bahwa tujuan utamanya adalah mengenai kata yang diucapkan tanpa harus tahu artinya, di mana bagian itu merupakan tugas “pemahaman suara”. Secara

umum prosesnya adalah usaha untuk menerjemahkan apa yang diucapkan seorang manusia menjadi kata-kata atau kalimat yang dapat dimengerti oleh komputer.

Sistem Sensor dan Robotika. Sistem sensor, seperti sistem visi dan pencitraan, serta sistem pengolahan sinyal, merupakan bagian dari robotika. Sebuah robot, yaitu perangkat elektromekanik yang diprogram untuk melakukan tugas manual, tidak semuanya merupakan bagian dari AI. Robot yang hanya melakukan aksi yang telah diprogramkan dikatakan sebagai robot bodoh yang tidak lebih pintar dari lift. Robot yang cerdas biasanya mempunyai perangkat sensor, seperti kamera, yang mengumpulkan informasi mengenai operasi dan lingkungannya. Kemudian bagian AI robot tersebut menerjemahkan informasi tadi dan merespon serta beradaptasi jika terjadi perubahan lingkungan.

Komputer Visi, merupakan kombinasi dari pencitraan, pengolahan citra, pengenalan pola serta proses pengambilan keputusan. Tujuan utama dari komputer visi adalah untuk menerjemahkan suatu pemandangan. Komputer visi banyak dipakai dalam kendali kualitas produk industri.

Intelligent Tutoring/Intelligent Computer-Aided Instruction, adalah komputer yang mengajari manusia. Belajar melalui komputer sudah lama digunakab, namun dengan menambahkan aspek kecerdasan di dalamnya, dapat tercipta komputer “guru” yang dapat mengatur teknik pengajarannya untuk menyesuaikan dengan kebutuhan “murid” secara individiual. Sistem ini juga mendukung pembelajaran bagi orang yang mempunyai kekurangan fisik atau kelemahan belajar.

Mesin Belajar (*Machine Learning*), yang berhubungan dengan sekumpulan metode untuk mencoba mengajari/melatih komputer untuk memecahkan masalah atau mendukung usaha pemecahan masalah dengan menganalisa kasus-kasus yang telah terjadi. Dua metode mesin belajar yang paling populer adalah **Komputasi Neural** dan **Logika Samar** yang akan dipelajari lebih dalam di bab-bab berikutnya.

Aplikasi lain dari AI misalnya untuk merangkum berita, pemrograman komputer secara otomatis, atau menerjemahkan dari suatu bahasa ke bahasa yang lain, serta aplikasi dalam permainan (Ingat pertandingan catur antara Grand Master Anatoly Karpov dengan komputer Deep Thought dari IBM).

BAB II
SISTEM PAKAR

A. Pendahuluan

Ketika hendak membuat suatu keputusan yang komplek atau memecahkan masalah, seringkali kita meminta nasehat atau berkonsultasi dengan seorang pakar atau ahli. Seorang **pakar** adalah seseorang yang mempunyai pengetahuan dan pengalaman spesifik dalam suatu bidang; misalnya pakar komputer, pakar uji tak merusak, pakar politik dan lain-lain. Semakin tidak terstruktur situasinya, semakin mengkhusus (dan mahal) konsultasi yang dibutuhkan.

Sistem Pakar (*Expert System*) adalah usaha untuk menirukan seorang pakar. Biasanya Sistem Pakar berupa perangkat lunak pengambil keputusan yang mampu mencapai tingkat performa yang sebanding seorang pakar dalam bidang problem yang khusus dan sempit. Ide dasarnya adalah: kepakaran ditransfer dari seorang pakar (atau sumber kepakaran yang lain) ke komputer, pengetahuan yang ada disimpan dalam komputer, dan pengguna dapat berkonsultasi pada komputer itu untuk suatu nasehat, lalu komputer dapat mengambil inferensi (menyimpulkan, mendeduksi, dll.) seperti layaknya seorang pakar, kemudian menjelaskannya ke pengguna tersebut, bila perlu dengan alasan-alasannya. Sistem Pakar malahan terkadang lebih baik unjuk kerjanya daripada seorang pakar manusia!

Kepakaran (*expertise*) adalah pengetahuan yang ekstensif (meluas) dan spesifik yang diperoleh melalui rangkaian pelatihan, membaca, dan pengalaman. Pengetahuan membuat pakar dapat mengambil keputusan secara lebih baik dan lebih cepat daripada non-pakar dalam memecahkan problem yang kompleks. Kepakaran mempunyai sifat berjenjang, pakar top memiliki pengetahuan lebih banyak daripada pakar yunior.

Tujuan Sistem Pakar adalah untuk mentransfer kepakaran dari seorang pakar ke komputer, kemudian ke orang lain (yang bukan pakar). Proses ini tercakup dalam rekayasa pengetahuan (*knowledge engineering*) yang akan dibahas kemudian.

B. Manfaat dan Keterbatasan Sistem Pakar

1. Manfaat Sistem Pakar

Mengapa Sistem Pakar menjadi sangat populer? Hal ini disebabkan oleh sangat banyaknya kemampuan dan manfaat yang diberikan oleh Sistem Pakar, di antaranya:

a. Meningkatkan output dan produktivitas, karena Sistem Pakar dapat bekerja lebih cepat dari manusia.
b. Meningkatkan kualitas, dengan memberi nasehat yang konsisten dan mengurangi kesalahan.
c. Mampu menangkap kepakaran yang sangat terbatas.
d. Dapat beroperasi di lingkungan yang berbahaya.
e. Memudahkan akses ke pengetahuan.
f. Handal. Sistem Pakar tidak pernah menjadi bosan dan kelelahan atau sakit. Sistem Pakar juga secara konsisten melihat semua detil dan tidak akan melewatkan informasi yang relevan dan solusi yang potensial.
g. Meningkatkan kapabilitas sistem terkomputerisasi yang lain. Integrasi Sistem Pakar dengan sistem komputer lain membuat lebih efektif, dan mencakup lebih banyak aplikasi .
h. Mampu bekerja dengan informasi yang tidak lengkap atau tidak pasti. Berbeda dengan sistem komputer konvensional, Sistem Pakar dapat bekerja dengan inofrmasi yang tidak lengkap. Pengguna dapat merespon dengan: “tidak tahu” atau “tidak yakin” pada satu atau lebih pertanyaan selama konsultasi, dan Sistem Pakar tetap akan memberikan jawabannya.
i. Mampu menyediakan pelatihan. Pengguna pemula yang bekerja dengan Sistem Pakar akan menjadi lebih berpengalaman. Fasilitas penjelas dapat berfungsi sebagai guru.
j. Meningkatkan kemampuan problem solving, karena mengambil sumber pengetahuan dari banyak pakar.
k. Meniadakan kebutuhan perangkat yang mahal.
l. Fleksibel.

2. Keterbatasan Sistem Pakar

Metodologi Sistem Pakar yang ada tidak selalu mudah, sederhana dan efektif. Berikut adalah keterbatasan yang menghambat perkembangan Sistem Pakar:

a. Pengetahuan yang hendak diambil tidak selalu tersedia.
b. Kepakaran sangat sulit diekstrak dari manusia.
c. Pendekatan oleh setiap pakar untuk suatu situasi atau problem bisa berbeda-beda, meskipun sama-sama benar.
d. Adalah sangat sulit bagi seorang pakar untuk mengabstraksi atau menjelaskan langkah mereka dalam menangani masalah
e. Pengguna Sistem Pakar mempunyai batas kognitif alami, sehingga mungkin tidak bisa memanfaatkan sistem secara maksimal.
f. Sistem Pakar bekerja baik untuk suatu bidang yang sempit.
g. Banyak pakar yang tidak mempunyai jalan untuk mencek apakah kesimpulan mereka benar dan masuk akal.
h. Istilah dan jargon yang dipakai oleh pakar dalam mengekspresikan fakta seringkali terbatas dan tidak mudah dimengerti oleh orang lain.
i. Pengembangan Sistem Pakar seringkali membutuhkan perekayasa pengetahuan (*knowledge engineer*) yang langka dan mahal.
j. Kurangnya rasa percaya pengguna menghalangi pemakaian Sistem Pakar.
k. Transfer pengetahuan dapat bersifat subyektif dan bias.

C. Komponen Sistem Pakar

Secara umum, Sistem Pakar biasanya terdiri atas beberapa komponen yang masing-masing berhubungan seperti terlihat pada Gambar II-1.

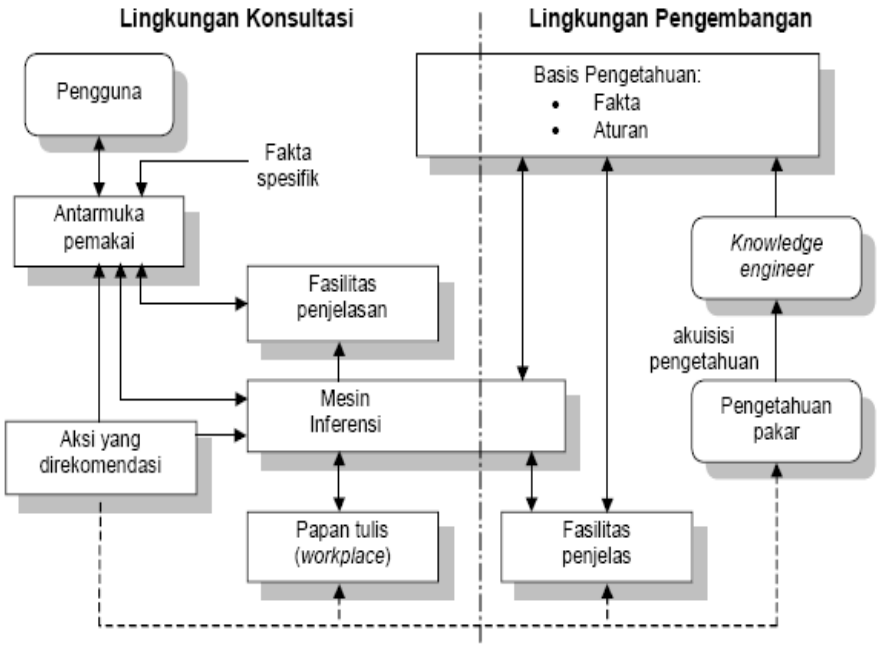
Basis Pengetahuan, berisi pengetahuan yang dibutuhkan untuk memahami, memformulasi, dan memecahkan masalah. Basis pengetahuan tersusun atas 2 elemen dasar:

1. Fakta, misalnya: situasi, kondisi, dan kenyataan dari permasalahan yang ada, serta teori dalam bidang itu
2. Aturan, yang mengarahkan penggunaan pengetahuan untuk memecahkan masalah yang spesifik dalam bidang yang khusus

Mesin Inferensi (*Inference Engine*), merupakan otak dari Sistem Pakar. Juga dikenal sebagai penerjemah aturan (*rule interpreter*). Komponen ini berupa program komputer yang menyediakan suatu metodologi untuk memikirkan (*reasoning*) dan memformulasi kesimpulan. Kerja mesin inferensi meliputi:

1. Menentukan aturan mana akan dipakai

2. Menyajikan pertanyaan kepada pemakai, ketika diperlukan.
3. Menambahkan jawaban ke dalam memori Sistem Pakar.
4. Menyimpulkan fakta baru dari sebuah aturan
5. Menambahkan fakta tadi ke dalam memori.



Gambar II-1. Struktur skematis sebuah Sistem Pakar

Papan Tulis (*Blackboard/Workplace*), adalah memori/lokasi untuk bekerja dan menyimpan hasil sementara. Biasanya berupa sebuah basis data.

Antarmuka Pemakai (*User Interface*). Sistem Pakar mengatur komunikasi antara pengguna dan komputer. Komunikasi ini paling baik berupa bahasa alami, biasanya disajikan dalam bentuk tanya-jawab dan kadang ditampilkan dalam bentuk gambar/grafik. Antarmuka yang lebih canggih dilengkapi dengan percakapan (*voice communication*).

Subsistem Penjelasan (*Explanation Facility*). Kemampuan untuk menjejak (*tracing*) bagaimana suatu kesimpulan dapat diambil merupakan hal yang sangat penting untuk transfer pengetahuan dan pemecahan masalah. Komponen subsistem penjelasan harus dapat menyediakannya yang secara interaktif menjawab pertanyaan pengguna, misalnya:

1. “Mengapa pertanyaan tersebut anda tanyakan?”
2. “Seberapa yakin kesimpulan tersebut diambil?”
3. “Mengapa alternatif tersebut ditolak?”

4. “Apa yang akan dilakukan untuk mengambil suatu kesimpulan?”
5. “Fakta apalagi yang diperlukan untuk mengambil kesimpulan akhir?”

Sistem Penghalusan Pengetahuan (*Knowledge Refining System*). Seorang pakar mempunyai sistem penghalusan pengetahuan, artinya, mereka bisa menganalisa sendiri performa mereka, belajar dari pengalaman, serta meningkatkan pengetahuannya untuk konsultasi berikutnya. Pada Sistem Pakar, swa-evaluasi ini penting sehingga dapat menganalisa alasan keberhasilan atau kegagalan pengambilan kesimpulan, serta memperbaiki basis pengetahuannya.

D. Pembangunan Sebuah Sistem Pakar

Mengembangkan Sistem Pakar dapat dilakukan dengan 2 cara:

1. Membangun sendiri semua komponen di atas, atau
2. Memakai semua komponen yang sudah ada kecuali isi basis pengetahuan.

Yang kedua disebut sebagai membangun Sistem Pakar dengan *shell*, yakni semua komponen Sistem Pakar, kecuali basis pengetahuan, bersifat generik; sehingga dapat dipakai untuk bidang yang berlainan. Membangun Sistem Pakar dengan *shell* dapat dilakukan dengan lebih cepat dan lebih sedikit keterampilan memprogram, namun berkurang fleksibilitasnya karena harus mengikuti kemampuan dari *shell* tersebut. Salah satu *shell* Sistem Pakar yang populer dipakai adalah CLIPS (*C Language Integrated Production System*) yang dapat di*download* dari internet.

1. Pemilihan Masalah

Pembuatan Sistem Pakar membutuhkan waktu dan biaya yang banyak. Untuk menghindari kegagalan yang memalukan dan kerugian yang besar, maka dibuat beberapa pedoman untuk menentukan apakah Sistem Pakar cocok untuk memecahkan suatu problem:

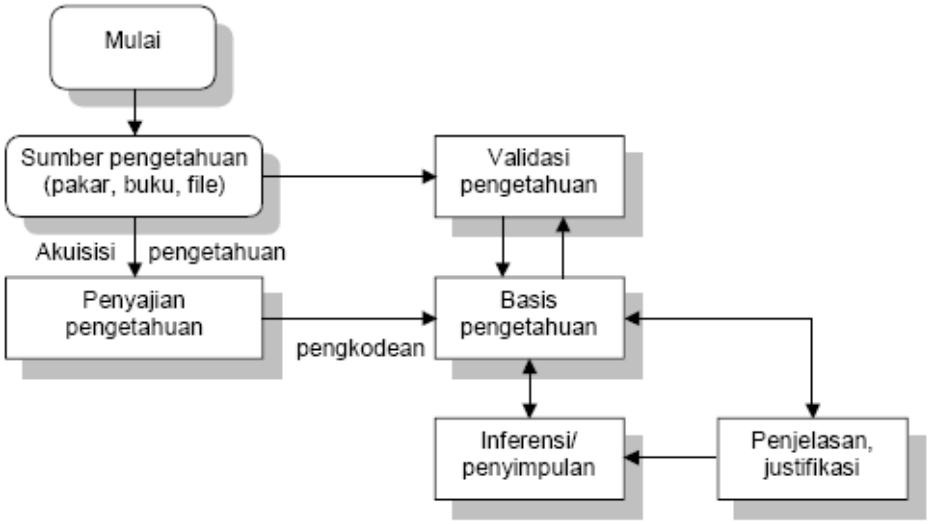
a. Biaya yang diperlukan untuk pembangunan Sistem Pakar ditentukan oleh kebutuhan untuk memperoleh solusi. Sehingga harus ada perhitungan yang realistis untuk *cost and benefit*.
b. Pakar manusia tidak mudah ditemui untuk semua situasi di mana dia dibutuhkan. Jika pakar pengetahuan tersebut terdapat di mana saja dan kapan saja, maka pembangunan Sistem Pakar menjadi kurang berharga.

c. Problem yang ada dapat diselesaikan dengan teknik penalaran simbolik, dan tidak membutuhkan kemampuan fisik.
d. Problem tersebut harus terstruktur dengan baik dan tidak membutuhkan terlalu banyak pengetahuan awam (*common sense*), yang terkenal sulit untuk diakuisisi dan dideskripsikan, dan lebih banyak berhubungan dengan bidang yang teknis.
e. Problem tersebut tidak mudah diselesaikan dengan metode komputasi yang lebih tradisionil. Jika ada penyelesaian algoritmis yang bagus untuk problem tersebut, maka kita tidak perlu memakai Sistem Pakar.
f. Ada pakar yang mampu memberikan penjelasan tentang kepakarannya serta mau bekerjasama. Adalah sangat penting bahwa pakar yang dihubungi benar-benar mempunyai kemauan kuat untuk ikut berpartisipasi serta tidak merasa pekerjaannya akan menjadi terancam.
g. Problem tersebut mempunyai sekup yang tepat. Biasanya merupakan problem yang membutuhkan kepakaran yang sangat khusus namun hanya membutuhkan seorang pakar untuk dapat menyelesaikannya dalam waktu yang relatif singkat (misalnya paling lama 1 jam).

2. Rekayasa Pengetahuan (*Knowledge Engineering*)

Proses dalam rekayasa pengetahuan meliputi (Gambar II-2):

a. Akuisisi pengetahuan, yaitu bagaimana memperoleh pengetahuan dari pakar atau sumber lain (sumber terdokumentasi, buku, sensor, file komputer, dll.).
b. Validasi pengetahuan, untuk menjaga kualitasnya misalnya dengan uji kasus.
c. Representasi pengetahuan, yaitu bagaimana mengorganisasi pengetahuan yang diperoleh, mengkodekan dan menyimpannya dalam suatu basis pengetahuan.
e. Penyimpulan pengetahuan, menggunakan mesin inferensi yang mengakses basis pengetahuan dan kemudian melakukan penyimpulan.
f. Transfer pengetahuan (penjelasan). Hasil inferensi berupa nasehat, rekomendasi, atau jawaban, kemudian dijelaskan ke pengguna oleh subsistem penjelas.



Gambar II-2. Proses dalam rekayasa pengetahuan

3. Partisipan Dalam Proses Pengembangan

Pakar, yaitu seseorang yang mempunyai pengetahuan, pengalaman, dan metode khusus, serta mampu menerapkannya untuk memecahkan masalah atau memberi nasehat. Pakar menyediakan pengetahuan tentang bagaimana nantinya Sistem Pakar bekerja.

Perekayasa pengetahuan (*knowledge engineer*), yang membantu pakar untuk menyusun area permasalahan dengan menerjemahkan dan mengintegrasikan jawaban pakar terhadap pertanyaan-pertanyaan dari klien, menarik analogi, serta memberikan contoh-contoh yang berlawanan, kemudian menyusun basis pengetahuan.

Pengguna, yang mungkin meliputi: seorang klien non-pakar yang sedang membutuhkan nasehat (Sistem Pakar sebagai konsultan atau *advisor*), seorang siswa yang sedang belajar (Sistem Pakar sebagai instruktur), seorang pembuat Sistem Pakar yang hendak meningkatkan basis pengetahuan (Sistem Pakar sebagai partner), seorang pakar (Sistem Pakar sebagai kolega atau asisten, yang dapat memberikan opini kedua).

Partisipan lain, dapat meliputi: pembangun sistem (*system builder*), *tool builder*, staf administrasi dsb.

4. Akuisisi Pengetahuan

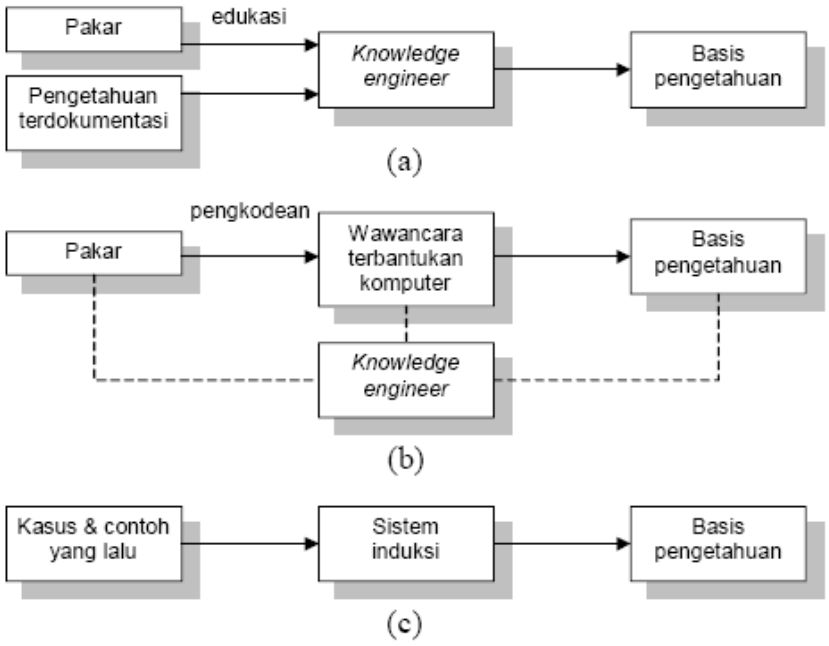
Dalam proses akuisisi pengetahuan, seorang perekayasa pengetahuan menjembatani antara pakar dengan basis pengetahuan. Perekayasa pengetahuan mendapatkan

pengetahuan dari pakar, mengolahnya bersama pakar tersebut, dan menaruhnya dalam basis pengetahuan, dengan format tertentu. Pengambilan pengetahuan dari pakar dapat dilakukan secara (Gambar II-3):

Manual, di mana perekayasa pengetahuan mendapatkan pengetahuan dari pakar (melalui wawancara) dan/atau sumber lain, kemudian mengkodekannya dalam basis pengetahuan. Proses ini biasanya berlangsung lambat, mahal, serta kadangkala tidak akurat.

Semi-otomatik, di mana terdapat peran komputer untuk: (1) mendukung pakar dengan mengijinkannya membangun basis pengetahuan tanpa (atau dengan sedikit) bantuan dari perekayasa pengetahuan, atau (2) membantu perekayasa pengetahuan sehingga kerjanya menjadi lebih efisien dan efektif.

Otomatik, di mana peran pakar, perekayasa pengetahuan, dan pembangun basis pengetahuan (*system builder*) digabung. Misalnya dapat dilakukan oleh seorang *system analyst* seperti pada metode induksi.



Gambar II-3. Metode akuisisi pengetahuan (a) manual (b) akuisisi terkendali-pakar (c) induksi

E. Representasi Pengetahuan

Setelah pengetahuan berhasil diakuisisi, mereka harus diorganisasi dan diatur dalam suatu konfigurasi dengan suatu format/representasi tertentu. Metode representasi pengetahuan yang populer adalah aturan produk dan bingkai.

1. Aturan Produk (*Production Rules*)

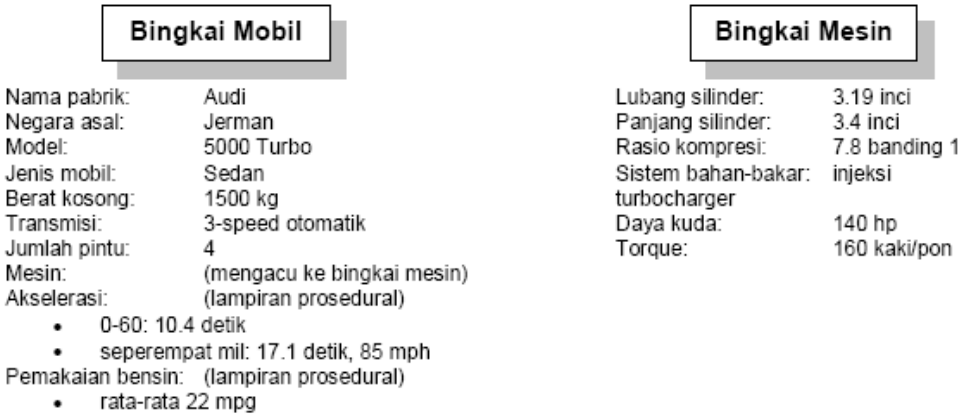
Di sini pengetahuan disajikan dalam aturan-aturan yang berbentuk pasangan keadaan-aksi (*condition-action*): "JIKA keadaan terpenuhi atau terjadi MAKA suatu aksi akan terjadi". Sistem Pakar yang basis pengetahuannya melulu disajikan dalam bentuk aturan produk disebut sistem berbasis-aturan (*rule-based system*). Kondisi dapat terdiri atas banyak bagian, demikian pula dengan aksi. Urutan keduanya juga dapat dipertukarkan letaknya. Contohnya:

a. JIKA suhu berada di bawah 20ºC MAKA udara terasa dingin.
b. Udara terasa dingin JIKA suhu berada di bawah 20ºC.
c. JIKA suhu berada di bawah 20ºC ATAU suhu berada di antara 20-25ºC DAN angin bertiup cukup kencang MAKA udara terasa dingin.
d. Contoh dari MYCIN, Sistem Pakar untuk mendiagnosis dan merekomendasikan perlakuan yang tepat untuk infeksi darah tertentu:

IF the infection is pimary-bacteremia
AND the site of the culture is one of the sterile sites
AND the suspected portal of entry is the gastrointestinal tract
THEN there is suggestive evidence (0.7) that infection is bacteroid.

2. Bingkai (*frame*)

Bingkai adalah struktur data yang mengandung semua informasi/pengetahuan yang relevan dari suatu obyek. Pengetahuan ini diorganisasi dalam struktur hirarkis khusus yang memungkinkan pemrosesan pengetahuan. Bingkai merupakan aplikasi dari pemrograman berorientasi obyek dalam AI dan Sistem Pakar. Pengetahuan dalam bingkai dibagi-bagi ke dalam slot atau atribut yang dapat mendeskripsikan pengetahuan secara deklaratif ataupun prosedural. Contoh frame untuk menggambarkan sebuah mobil diberikan dalam Gambar II-4 berikut ini.



Gambar II-4. Bingkai pengetahuan untuk sebuah mobil

Contoh lain adalah bingkai untuk merepresentasikan pengetahuan mengenai gajah:

```
Mamalia
subkelas: Binatang
berdarah_panas: ya
Gajah
subkelas: Mamalia
* warna: abu-abu
* ukuran: besar
Clyde
instance: Gajah
warna: merah_muda
pemilik: Fred
Nellie:
instance: Gajah
ukuran: kecil
```

F. Bagaimana Sistem Pakar Melakukan Inferensi?

1. Sistem Perantaian Maju (*Forward Chaining Systems*)

Pada sistem perantaian maju, fakta-fakta dalam dalam sistem disimpan dalam memori kerja dan secara kontinyu diperbarui. Aturan dalam sistem merepresentasikan aksi-aksi yang harus diambil apabila terdapat suatu kondisi khusus pada item-item dalam memori kerja, sering disebut aturan kondisi-aksi. Kondisi biasanya berupa pola yang cocok dengan item yang ada di dalam memori kerja, sementara aksi biasanya berupa penambahan atau penghapusan item dalam memori kerja.

Aktivitas sistem dilakukan berdasarkan siklus mengenal-beraksi (*recognise-act*). Mula-mula, sistem mencari semua aturan yang kondisinya terdapat di memori kerja, kemudian memilih salah satunya dan menjalankan aksi yang bersesuaian dengan aturan tersebut. Pemilihan aturan yang akan dijalankan (*fire*) berdasarkan strategi tetap yang disebut strategi penyelesain konflik. Aksi tersebut menghasilkan memori kerja baru, dan siklus diulangi lagi sampai tidak ada aturan yang dapat dipicu (*fire*), atau *goal* (tujuan) yang dikehendaki sudah terpenuhi.

Sebagai contoh, lihat pada sekumpulan aturan sederhana berikut (Di sini kita memakai kata yang diawali huruf kapital untuk menyatakan suatu variabel. Pada sistem lain, mungkin dipakai cara lain, misalnya menggunakan awalan ? atau ^):

```
1. JIKA (mengajar X) DAN (mengoreksi_tugas X) MAKA TAMBAH
   (terlalu_banyak_bekerja X)
2. JIKA (bulan maret) MAKA TAMBAH (mengajar balza)
3. JIKA (bulan maret) MAKA TAMBAH (mengoreksi_tugas balza)
4. JIKA (terlalu_banyak_bekerja X) ATAU (kurang_tidur X) MAKA
   TAMBAH (mood_kurang_baik X)
5. JIKA (mood_kurang_baik X) MAKA HAPUS (bahagia X)
6. JIKA (mengajar X) MAKA HAPUS (meneliti X)
```

Kita asumsikan, pada awalnya kita mempunyai memori kerja yang berisi fakta berikut:

```
(bulan maret)
(bahagia balza)
(meneliti balza)
```

Sistem Pakar mula-mula akan memeriksa semua aturan yang ada untuk mengenali aturan manakah yang dapat memicu aksi, dalam hal ini aturan 2 dan 3. Sistem kemudian memilih

salah satu di antara kedua aturan tersebut dengan strategi penyelesaian konflik. Katakanlah aturan 2 yang terpilih, maka fakta `(mengajar balza)` akan ditambahkan ke dalam memori kerja. Keadaan memori kerja sekarang menjadi:

```
(mengajar balza)
(bulan maret)
(bahagia balza)
(meneliti balza)
```

Sekarang siklus dimulai lagi, dan kali ini aturan 3 dan 6 yang kondisinya terpenuhi. Katakanlah aturan 3 yang terpilih dan terpicu, maka fakta `(mengoreksi_tugas balza)` akan ditambahkan ke dalam memori kerja. Lantas pada siklus ketiga, aturan 1 terpicu, sehingga variabel X akan berisi (*bound to*) balza, dan fakta `(terlalu_banyak_bekerja balza)` ditambahkan, sehingga isi memori kerja menjadi:

```
(terlalu_banyak_bekerja balza)
(mengoreksi_tugas balza)
(mengajar balza)
(bulan maret)
(bahagia balza)
(meneliti balza)
```

Aturan 4 dan 6 dapat diterapkan. Misalkan aturan 4 yang terpicu, sehingga fakta `(mood_kurang_baik balza)` ditambahkan. Pada siklus berikutnya, aturan 5 terpilih dan dipicu, sehingga fakta `(bahagia balza)` dihapus dari memori kerja. Kemudian aturan 6 akan terpicu dan fakta `(meneliti balza)` dihapus pula dari memori kerja menjadi:

```
(mood_kurang_baik balza)
(terlalu_banyak_bekerja balza)
(mengoreksi_tugas balza)
(mengajar balza)
(bulan maret)
```

Urutan aturan yang dipicu bisa jadi sangat vital, terutama di mana aturan-aturan yang ada dapat mengakibatkan terhapusnya item dari memori kerja. Tinjau kasus berikut: andaikan terdapat tambahan aturan pada kumpulan aturan di atas, yaitu:

```
7. JIKA (bahagia X) MAKA TAMBAH (memberi_nilai_bagus X)
```

Jika aturan 7 ini terpicu sebelum `(bahagia balza)` dihapus dari memori, maka Sistem Pakar akan berkesimpulan bahwa saya akan memberi nilai bagus[(1)]. Namun jika aturan 5 terpicu dahulu, maka aturan 7 tidak akan dijalankan (artinya saya tidak akan memberi nilai bagus).

2. Strategi penyelesaian konflik (*conflict resolution strategy*)

Strategi penyelesaian konflik dilakukan untuk memilih aturan yang akan diterapkan apabila terdapat lebih dari 1 aturan yang cocok dengan fakta yang terdapat dalam memori kerja. Di antaranya adalah:

a. *No duplication*. Jangan memicu sebuah aturan dua kali menggunakan fakta/data yang sama, agar tidak ada fakta yang ditambahkan ke memori kerja lebih dari sekali.
b. *Recency*. Pilih aturan yang menggunakan fakta yang paling baru dalam memori kerja. Hal ini akan membuat sistem dapat melakukan penalaran dengan mengikuti rantai tunggal ketimbang selalu menarik kesimpulan baru menggunakan fakta lama.
c. *Specificity*. Picu aturan dengan fakta prakondisi yang lebih spesifik (khusus) sebelum aturan yang mengunakan prakondisi lebih umum. Contohnya: jika kita mempunyai aturan "JIKA (burung X) MAKA TAMBAH (dapat_terbang X)" dan "JIKA (burung X) DAN (pinguin X) MAKA TAMBAH (dapat_berenang X)" serta fakta bahwa tweety adalah seekor pinguin, maka lebih baik memicu aturan kedua dan menarik kesimpulan bahwa tweety dapat berenang.
d. *Operation priority*. Pilih aturan dengan prioritas yang lebih tinggi. Misalnya ada fakta (bertemu kambing), (ternak kambing), (bertemu macan), dan (binatang_buas macan), serta dua aturan: "JIKA (bertemu X) DAN (ternak X) MAKA TAMBAH (memberi_makan X)" dan "JIKA (bertemu X) DAN (binatang_buas X) MAKA TAMBAH (melarikan_diri)", maka kita akan memilih aturan kedua karena lebih tinggi prioritasnya.

3. Sistem Perantaian Balik (*Backward Chaining Systems*)

Sejauh ini kita telah melihat bagaimana sistem berbasis aturan dapat digunakan untuk menarik kesimpulan baru dari data yang ada, menambah kesimpulan ini ke dalam memori kerja. Pendekatan ini berguna ketika kita mengetahui semua fakta awalnya, namun tidak dapat menebak konklusi apa yang bisa diambil. Jika kita tahu kesimpulan apa yang seharusnya, atau mempunyai beberapa hipotesis yang spesifik, maka perantaian maju di atas menjadi tidak efisien. Sebagai contoh, jika kita ingin mengetahui apakah saya dalam keadaan mempunyai mood yang baik sekarang, kemungkinan kita akan berulangkali memicu aturan-aturan dan memperbarui memori kerja untuk mengambil kesimpulan apa yang terjadi pada bulan Maret, atau apa yang terjadi jika saya mengajar, yang sebenarnya

perlu terlalu kita ambil pusing. Dalam hal ini yang diperlukan adalah bagaimana dapat menarik kesimpulan yang relevan dengan tujuan atau *goal*.

Hal ini dapat dikerjakan dengan perantaian balik dari pernyataan goal (atau hipotesis yang menarik bagi kita). Jika diberikan sebuah goal yang hendak dibuktikan, maka mula-mula sistem akan memeriksa apakah goal tersebut cocok dengan fakta-fakta awal yang dimiliki. Jika ya, maka goal terbukti atau terpenuhi. Jika tidak, maka sistem akan mencari aturan-aturan yang konklusinya (aksinya) cocok dengan goal. Salah satu aturan tersebut akan dipilih, dan sistem kemudian akan mencoba membuktikan fakta-fakta prakondisi aturan tersebut menggunakan prosedur yang sama, yaitu dengan menset prakondisi tersebut sebagai goal baru yang harus dibuktikan.

Perhatikan bahwa pada perantaian balik, sistem tidak perlu memperbarui memori kerja, namun perlu untuk mencatat goal-goal apa saja yang dibuktikan untuk membuktikan goal utama (hipotesis).

Secara prinsip, kita dapat menggunakan aturan-aturan yang sama untuk perantaian maju dan balik. Namun, dalam prakteknya, harus sedikit dimodifikasi. Pada perantaian balik, bagian MAKA dalam aturan biasanya tidak diekspresikan sebagai suatu aksi untuk dijalankan (misalnya TAMBAH atau HAPUS), tetapi suatu keadaan yang bernilai benar jika premisnya (bagian JIKA) bernilai benar. Jadi aturan-aturan di atas diubah menjadi:

1. `JIKA (mengajar X) DAN (mengoreksi_tugas X) MAKA (terlalu_banyak_bekerja X)`
2. `JIKA (bulan maret) MAKA (mengajar balza)`
3. `JIKA (bulan maret) MAKA (mengoreksi_tugas balza)`
4. `JIKA (terlalu_banyak_bekerja X) ATAU (kurang_tidur X) MAKA (mood_kurang_baik X)`
5. `JIKA (mood_kurang_baik X) MAKA TIDAK BENAR (bahagia X)`

dengan fakta awal:

```
(bulan maret)
(meneliti balza)
```

Misalkan kita hendak membuktikan apakah mood saya sedang kurang baik. Mula-mula kita periksa apakah goal cocok dengan fakta awal. Ternyata tidak ada fakta awal yang menyatakan demikian, sehingga langkah kedua yaitu mencari aturan mana yang mempunyai konklusi `(mood_kurang_baik balza)`. Dalam hal ini aturan yang cocok adalah aturan 4 dengan variabel X diisi dengan (*bound to*) `balza`. Dengan demikian kita harus membuktikan bahwa prakondisi aturan ini, `(terlalu_banyak_bekerja balza)`

atau `(kurang_tidur balza)`, salah satunya adalah benar (karena memakai ATAU). Lalu diperiksa aturan mana yang dapat membuktikan bahwa adalah `(terlalu_banyak_bekerja balza)` benar, ternyata aturan 1, sehingga prakondisinya, `(mengajar X)` dan `(mengoreksi_tugas X)`, dua-duanya adalah benar (karena memakai DAN). Ternyata menurut aturan 2 dan 3, keduanya bernilai bernilai benar jika `(bulan maret)` adalah benar. Karena ini sesuai dengan fakta awal, maka keduanya bernilai benar. Karena semua goal sudah terpenuhi maka goal utama (hipotesis) bahwa mood saya sedang kurang baik adalah benar (terpenuhi).

Untuk mencatat goal-goal yang harus dipenuhi/dibuktikan, dapat digunakan *stack* (tumpukan). Setiap kali ada aturan yang konklusinya cocok dengan goal yang sedang dibuktikan, maka fakta-fakta prakondisi dari aturan tersebut ditaruh (*push*) ke dalam stack sebagai goal baru. Dan setiap kali goal pada tumpukan teratas terpenuhi atau dapat dibuktikan, maka goal tersebut diambil (*pop*) dari tumpukan. Demikian seterusnya sampai tidak ada goal lagi di dalam *stack*, atau dengan kata lain goal utama (yang terdapat pada tumpukan terbawah) sudah terpenuhi.

4. Pemilihan Sistem Inferensi

Secara umum kita dapat memakai panduan berikut untuk menentukan apakah kita hendak memilih perantaian maju atau balik untuk Sistem Pakar yang kita bangun. Panduan tersebut tercantum dalam Tabel II-1 berikut ini.

Tabel II-1. Panduan untuk memilih sistem inferensi

Perantaian Maju	Perantaian Balik
• Ada banyak hal yang hendak dibuktikan	• Hanya akan membuktikan fakta (hipotesis) tunggal
• Hanya sedikit fakta awal yang dipunyai	• Terdapat banyak fakta awal
• Ada banyak aturan berbeda yang dapat memberikan kesimpulan yang sama	• Jika terdapat banyak aturan yang memenuhi syarat untuk dipicu (*fire*) pada suatu siklus

5. Ketidakpastian dalam Aturan

Sejauh ini kita menggunakan nilai kebenaran tegas dalam fakta dan aturan yang dipakai, misalnya: jika terlalu banyak bekerja maka pasti mood kurang baik. Pada kenyataanya, seringkali kita tidak bisa membuat aturan yang absolut untuk mengambil kesimpulan secara pasti, misalnya: jika terlalu banyak bekerja maka kemungkinan besar mood kurang baik. Untuk itu, seringkali aturan yang dipakai memiliki nilai kepastian (*certainty value*). Contohnya: jika terlalu banyak bekerja maka pasti mood kurang baik (kepastian 0,75).

G. Contoh Aplikasi Sistem Pakar

1. Aplikasi Sederhana: Sistem Pakar Bengkel Mobil

Ini adalah contoh Sistem Pakar sederhana, yang bertujuan untuk mencari apa yang salah sehingga mesin mobil pelanggan yang tidak mau hidup, dengan memberikan gejala-gejala yang teramati. Anggap Sistem Pakar kita memiliki aturan-aturan berikut:

1. `JIKA mesin_mendapatkan_bensin DAN starter_dapat_dihidupkan MAKA ada_masalah_dengan_pengapian`
2. `JIKA TIDAK BENAR starter_dapat_dihidupkan DAN TIDAK BENAR lampu_menyala MAKA ada_masalah_dengan_aki`
3. `JIKA TIDAK BENAR starter_dapat_dihidupkan DAN lampu_menyala MAKA ada_masalah_dengan_starter`
4. `JIKA ada_bensin_dalam_tangki_bahan_bakar MAKA mesin_mendapatkan_bensin`

Terdapat 3 masalah yang mungkin, yaitu: `ada_masalah_dengan_pengapian`, `ada_masalah_dengan_aki` dan `ada_masalah_dengan_starter`. Dengan sistem terarah-tujuan (*goal-driven*), kita hendak membuktikan keberadaan setiap masalah tadi.

Pertama, Sistem Pakar berusaha untuk membuktikan kebenaran `ada_masalah_dengan_pengapian`. Di sini, aturan 1 dapat digunakan, sehingga Sistem Pakar akan menset goal baru untuk membuktikan apakah `mesin_mendapatkan_bensin` serta `starter_dapat_dihidupkan`. Untuk membuktikannya, aturan 4 dapat digunakan, dengan goal baru untuk membuktikan `mesin_mendapatkan_bensin`. Karena tidak ada aturan lain yang dapat digunakan menyimpulkannya, sedangkan sistem belum memperoleh

solusinya, maka Sistem Pakar kemudian bertanya kepada pelanggan: "*Apakah ada bensin dalam tangki bahan bakar?*". Sekarang, katakanlah jawaban klien adalah "*Ya*", jawaban ini kemudian dicatat, sehingga klien tidak akan ditanyai lagi dengan pertanyaan yang sama.

Nah, karena sistem sekarang sudah dapat membuktikan bahwa mesin mendapatkan bensin, maka sistem sekarang berusaha mengetahui apakah `starter_dapat_dihidupkan`. Karena sistem belum tahu mengenai hal ini, sementara tidak ada aturan lagi yang dapat menyimpulkannya, maka Sistem Pakar bertanya lagi ke klien: "*Apakah starter dapat dihidupkan?*". Misalkan jawabannya adalah "*Tidak*", maka tidak ada lagi aturan yang dapat membuktikan `ada_masalah_dengan_pengapian`, sehingga Sistem Pakar berkesimpulan bahwa hal ini bukanlah solusi dari problem yang ada, dan kemudian melihat hipotesis berikutnya: `ada_masalah_dengan_aki`. Sudah diketahui (dibuktikan) bahwa mesin tidak dapat distarter, sehingga yang harus dibuktikan adalah bahwa lampu tidak menyala. Sistem Pakar kemudian bertanya: "*Apakah lampu menyala?*". Misalkan jawabannya adalah "*Tidak*", maka sudah terbukti bahwa ada masalah dengan aki.

Sistem ini mungkin berhenti sampai di sini, tetapi biasanya ada kemungkinan terdapat lebih dari satu solusi (misalnya terdapat lebih dari satu kerusakan), atau ada kemungkinan terdapat solusi lain yng lebih tepat, sehingga biasanya semua hipotesis diperiksa kebenarannya. Sistem Pakar ini kemudian mencoba membuktikan bahwa `ada_masalah_dengan_starter`, namun dari fakta yang sudah diperoleh, yaitu lampu tidak menyala, maka pembuktiannya menjadi gagal. Dengan demikian solusi yang diberikan oleh Sistem Pakar adalah ada masalah dengan aki.

Secara lengkap, interaksi antara Sistem Pakar dengan klien mungkin seperti ini:

System Pakar: *Apakah ada bensin dalam tangki bahan bakar?*

Klien: Ya

System Pakar: *Apakah starter dapat dihidupkan?*

Klien: Tidak

System Pakar: *Apakah lampu menyala?*

Klien: Tidak

System Pakar: *Saya berkesimpulan bahwa ada masalah dengan aki*

Fasilitas Penjelasan

Kebanyakan Sistem Pakar mempunyai fasilitas penjelasan yang mengijinkan klien untuk bertanya mengapa dia diberi suatu pertanyaan atau bagaimana dapat mencapai suatu kesimpulan. Pertanyaan dari klien tersebut dijawab dengan mengacu pada goal, aturan yang dipakai, dan data-data yang ada. Contoh pemakaian fasilitas penjelas dalam Sistem Pakar di atas dapat dilihat pada percakapan berikut:

System Pakar: *Apakah ada bensin dalam tangki bahan bakar?*

Klien: Ya

System Pakar: *Apakah starter dapat dihidupkan?*

Klien: Tidak

System Pakar: *Apakah lampu menyala?*

Klien: Mengapa?

System Pakar: *Saya mencoba membuktikan bahwa ada masalah dengan aki. Sudah ditunjukkan adalah tidak benar starter dapat dihidupkan. Sehingga jika tidak benar bahwa lampu menyala, maka ada masalah dengan aki. Apakah lampu menyala?*

Klien: Tidak

System Pakar: *Saya berkesimpulan bahwa ada masalah dengan aki*

Klien: Bagaimana bisa?

System Pakar: *Hal ini sesuai dengan aturan no 2: JIKA TIDAK BENAR starter_dapat_dihidupkan DAN TIDAK BENAR lampu_menyala MAKA ada_masalah_dengan_aki bahwa tidak benar starter_dapat_dihidupkan sudah dijawab, bahwa tidak benar lampu_menyala sudah dijawab, sehingga dapat disimpulkan bahwa ada_masalah_dengan_aki.*

Fasilitas penjelasan berhubungan dengan pencatatan aturan apa saja yang digunakan untuk mengambil kesimpulan, dan menggunakan catatan tersebut untuk menyusun penjelasan. Pembangkitan penjelasan sederhana seperti contoh di atas bukanlah persoalan yang sulit, namun pemahaman klien terhadap penjelasan tersebut tidaklah sama, sehingga seringkali klien tetap tidak mengerti maksudnya. Hal ini karena biasanya penjelasan hanya meliputi pengetahuan yang dangkal atau hanya permukaan saja sebagaimana tercantum dalam autran yang dipunyai, bukannya pengetahuan yang mendalam yang mendasari aturan tersebut. Dalam contoh di atas, Sistem Pakar hanya

mengatakan bahwa hal ini sesuai dengan aturan no 2, tetapi tidak menjelaskan secara lebih detil mengenai aturan 2 tersebut. Mungkin klien perlu diberi pengertian bahwa lampu dan starter dua-duanya menggunakan aki untuk menyalakannya. Hal penting lain adalah ketika klien gagal mengerti atau menerima penjelasan yang diberikan, biasanya Sistem Pakar tidak dapat memberi penjelasan dengan cara lain (tidak begitu halnya dengan seorang pakar yang mampu menggunakan kalimat yang lain untuk mencoba menjelaskannya).

2. Eliza

Eliza adalah salah satu Sistem Pakar yang paling awal dikembangkan. Ini adalah program komputer terapis yang dibuat oleh Joseph Weizenbaum di MIT. Pengguna berkomunikasi dengannya sebagaimana sedang berkonsultasi dengan seorang terapis. Berikut ini adalah contoh percakapan antara Eliza dengan seorang klien:

Eliza: *Hello. I am ELIZA. How can I help you?*

Klien: My name is Balza.

Eliza: *I have told you before, I do not care about names.*

Klien: I don’t have any money. That’s my problem.

Eliza: *Why do you say your problem?*

Klien: Because you asked me.

Eliza: *Do any other reasons not come to mind?*

Klien: No.

Eliza: *You are being a bit negative.*

Klien: Why?

Eliza: *Please go on.*

Klien: OK. I am bankrupt.

Eliza: *How long have you been bankrupt?*

Klien: A year.

Eliza: *Please go on.*

Klien: So, what should I do?

Eliza: *Does that question interest you?*

Klien: Yes, how can I make big money?

Eliza: *You are sure?*

Klien: Of course I’m sure.

...

3. Parry

Parry adalah Sistem Pakar yang juga paling awal dikembangkan di Stanford University oleh seorang psikiater, Kenneth Colby, yang mensimulasikan seorang paranoid. Berikut ini contoh respon Parry terhadap pertanyaan seorang psikiater yang menginterviunya.

Psikiater: Why are you in the hospital?

Parry: *I shouldn't be here.*

Psikiater: Who brought you here?

Parry: *The police.*

Psikiater: What trouble did you have with the police?

Parry: *Cops don't do their job.*

Psikiater: What do they do instead?

Parry: *Cops arrest the wrong people.*

Psikiater: They arrest the right ones sometimes

Parry: *The law should get those Italian crooks*.

Psikiater: Are you Italian?

Parry: *My background is British but I was born in this country.*

Psikiater: Do you know any Italian crooks?

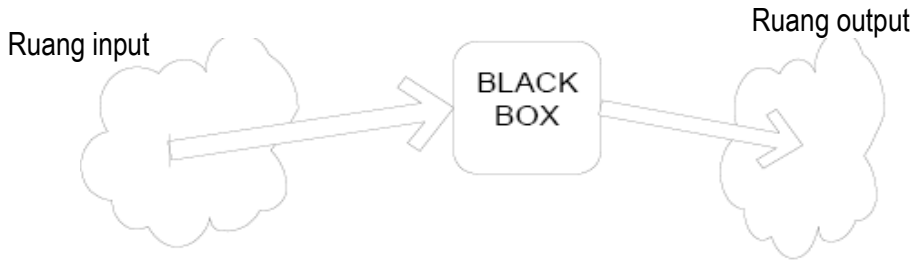
Parry: *I try to avoid the underworld.*

...

BAB III
LOGIKA FUZZY
(*Review Fuzzy Logic Controls*)

Model Fuzzy Logik

Pada dasarnya teknik kontrol standar adalah menggunakan data kuantitatip untuk menghubungkan sinyal masukan dan keluaran. Sedangkan sistem Fuzzy Logik, dapat menggunakan kedua-dua informasi kuantitatip dan informasi linguistik untuk memetakan. Diagram berikut menggambarkan pemikiran tadi [9]



Gambar pemetaan ruang masukan ke ruang keluaran

Beraneka mekanisme kontrol bisa berada di dalam kotak hitam, tetapi dalam hal ini mekanisme yang yang akan dibahas suatu sistim Fuzzy Logik. Karena sasaran itu untuk memetakan masukan ke keluaran yang mungkin untuk model sistem takliniar, bahkan komplek. Ini adalah keunggulan utama Fuzzy Logik. Sistem ini juga dapat beradaptasi dengan baik pada sistem yang berbeda, karena sistem Fuzzy Logik bersifat toleran terhadap data yang tidak tepat. Sistem kontrol ini dapat melakukan banyak penyesuaian dengan baik walaupun sistem dunia nyata terus meningkat komplek. Sering kali kebutuhan data tepat, berkurang, seiring dengan meningkatnya kompleksitas sistem. Kaidah pemetaan masukan-keluaran ini dapat diperoleh melalui dua metoda. Pertama adalah metoda pendekatan langsung dan yang kedua adalah identifikasi sistim. Pendekatan langsung melibatkan pakar manusia untuk memformulasikan kaidah-kaidah linguistik. Kaidah-kaidah ini kemudian diubah menjadi model formal sistem Fuzzy Logik. Yang menjadi masalah dengan pendekatan ini adalah sedikit sekali pakar yang mengenal sistem dengan baik, juga sangat sulit untuk merancang suatu base kaidah Fuzzy Logik dan sistim inference yang dapat bekerja sendiri secara efisien. Untuk sistem kompleks (taklinear)

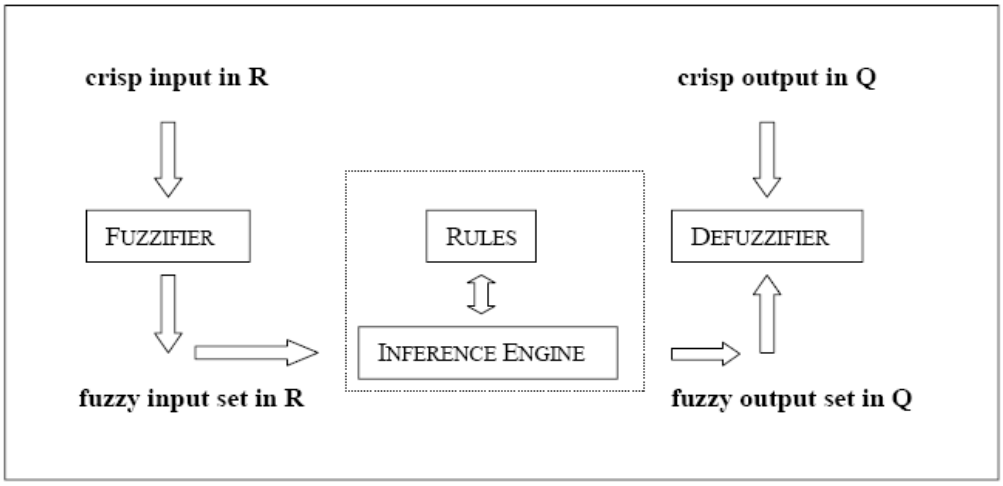
kesulitan terletak pada *tuning* terhadap fungsi keanggotaan dan banyak parameter lain harus dilakukan secara simultan.

Model Fuzzy Logik yang dirancang dengan identifikasi sistim didasarkan pada pemakaian data input-output. Identifikasi sistim mengatasi kesulitan yang dihadapi pendekatan langsung tentang memilih fungsi-fungsi keanggotaan himpunan Fuzzy Logik, menggunakan teknik search/optimisasi untuk membantu pemilihan. Teknik ini adalah Algoritma Genetik.

Semua unsur Fuzzy Logik yang telah dibahas dirangkumkan untuk membentuk sistim kesimpulan yang disebut Fuzzy Inference System (FIS). Dua tipe utama dari FIS adalah tipe Mamdani dan Sugeno.

Model Mamdani

Tipe pertama dikembangkan oleh Ebrahim Mamdani. Model kontrol Fuzzy pertama yang efisien ini dirancang dan diperkenalkan pada tahun 1975 [10]. Kontrol Fuzzy terdiri dari fuzzifier, Fuzzy Logik rule-base, inference engine dan defuzzifier.



Sistem kontrol konvensional memerlukan keluaran analog *crisp* dari masukan *crisp*. Gambar tersebut menunjukkan bagaimana suatu masukan *crisp* dalam R dapat dioperasikan oleh sistim Fuzzy Logik untuk menghasilkan keluaran *crisp* dalam Q. Kontrol tipe Mamdani disadari langkah-langkah berikut.

A. Fuzzification Masukan-masukan

B. Penerapan Operator Fuzzy Logik

C. Penerapan metoda implikasi

D. Pengumpulan Keluaran

E. Defuzzification Keluaran

A. Fuzzification Masukan-masukan

Fuzzifier memetakan angka-angka masukan *crisp* ke dalam himpunan Fuzzy Logik. Nilai masukan digantikan oleh nilai fuzzy antara 0 dan 1. Nilai fuzzy ini menunjukkan derajat keanggotaan masukan. Pemetaaan keluaran juga dengan cara ini, yaitu mempunyai derajad keanggotaan dengan nilai antara 0 dan 1. Fuzzification dapat diterapkan dengan tabel *look-up* atau seperti di dalam usulan ini, menggunakan *membership function*.

B. Penerapan Operator Fuzzy Logik

Pada kasus di mana pernyataan ganda digunakan dalam *antecedent* aturan, perlu menerapkan operator Fuzzy Logik dengan benar. Operator and dan or sering dipakai dalam Fuzzy Logik tipe Mamdani. Operator dalam antecedent aturan memungkinkan pernyataan diwakili oleh satu bilangan yang berarti sekaligus menunjukkan kekuatan kaidah ini.

C. Penerapan metoda implikasi

Sistim Mamdani melukiskan konsekuensi sebagai satu himpunan keluaran Fuzzy Logik. Konsekuen hanya dapat dicapai setelah masing-masing antecedent sudah dievaluasi dan ditentukan bobot-nya untuk menentukan himpunan keluaran Fuzzy Logik.

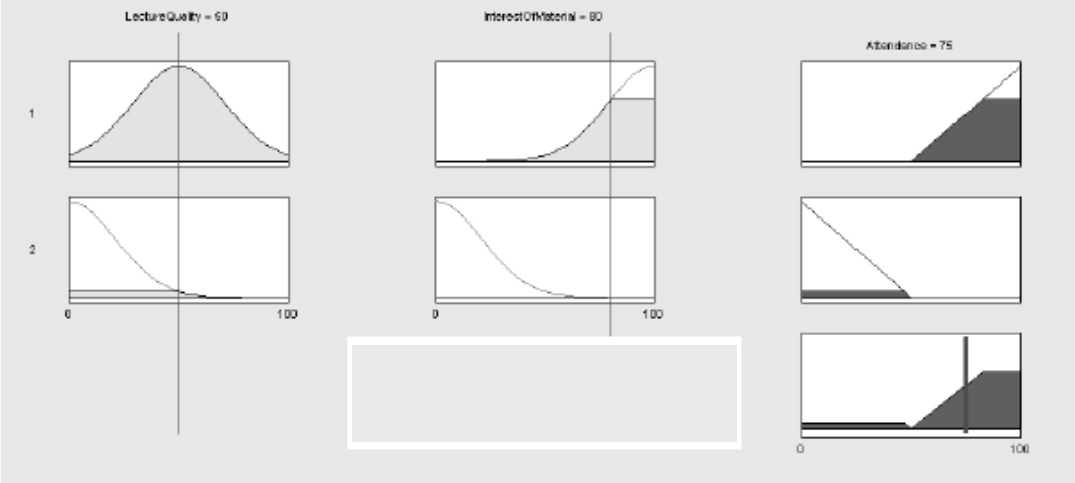
D. Pengumpulan semua Keluaran

Keluaran Fuzzy Logik dari tiap aturan perlu dikombinasikan agar mempunyai arti dan berguna. Pengumpulan adalah metoda yang digunakan untuk melaksanakan hal ini yaitu dengan kombinasi masing-masing keluaran Fuzzy Logik. Metoda-metoda pengumpulan yang sering dipakai untuk tipe Mamdani adalah jumlahan (*sum*), himpunan keluaran adalah jumlah dari tiap aturan, max (nilai maksimum dari semua keluaran aturan) dan metoda OR probabilistic (jumlah aljabar dari tiap keluaran aturan). Satu contoh dari proses pengumpulan yang menggunakan operator max dapat dilihat pada Gambar 4 di bawah.

E. Defuzzification Keluaran

Himpunan Fuzzy Logik yang dikumpulkan menjadi masukan kepada defuzzifier. Seperti ditunjukkan dalam Gambar, hasil kumpulan Fuzzy Logik yang ditetapkan dalam Q

dan *crisp* dalam Q menjadi masukan bagi defuzzifier. Keluaran *crisp* ini adalah suatu bilangan berguna untuk mengendalikan sistim. Sejumlah metoda-metoda defuzzification antara lain *mean of maximum, largest of maximum, smallest of maximum and centroid (centre of area)*. Metode sentroid itu adalah paling luas digunakan dan dapat dilihat pada Gambar 4.



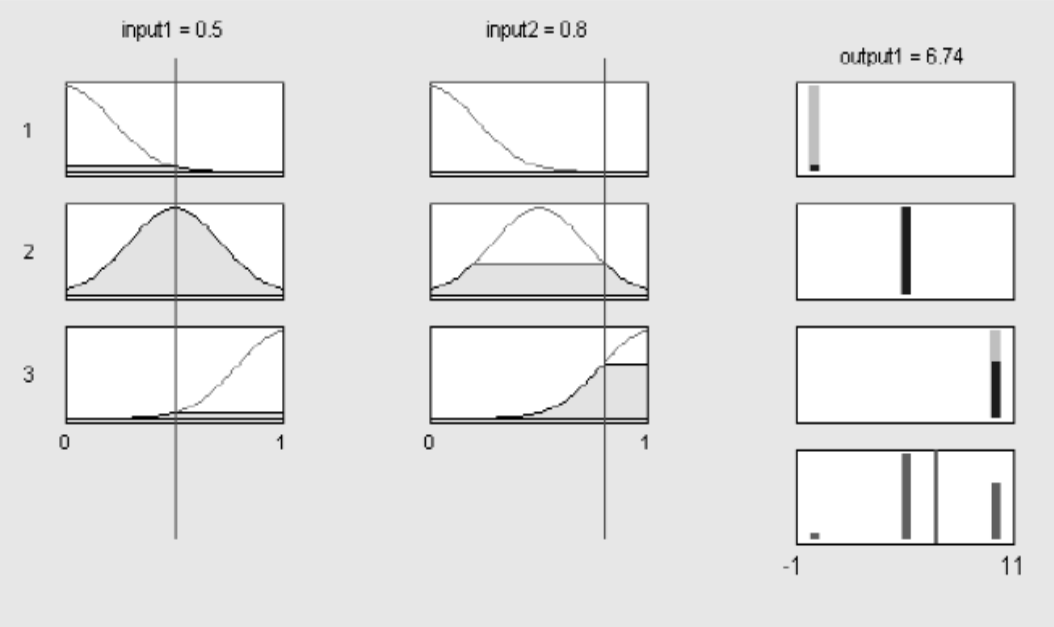
Gambar 4 Diagram yang mempertunjukkan pengumpulan dan defuzzification

Model Sugeno

Model Sugeno Fuzzy Logik, atau lebih lengkap metoda Takagi-Sugeno-Kang, adalah FIS pertamakali diperkenalkan pada tahun 1985 [10]. Dalam banyak hal model ini serupa dengan metoda Mamdani kecuali bahwa *membership-function* keluaran metoda Sugeno selalu linear atau konstan. *Membership-function* mudah dimengerti sebagai suatu pengumpulan yang sederhana daripada metoda-metoda pengumpulan yang lain seperti *max, probor* atau *sum*. Gambar 5 menunjukkan penerapan dari tiga ketentuan dasar untuk suatu model Sugeno. Ketiga kaidah tersebut telah ditulis dengan konektor OR. Sebagai contoh:

if input1 is x or input2 is y then output1 is z

Metoda defuzzification adalah rata-rata tertimbang. Keluaran itu selalu suatu konstanta. Sistim Sugeno melakukan komputasi secara efisien dan kemampuannya untuk menyisipkan model multiple-linear membuatnya cocok untuk model sistem taklinear.



Gambar Implementasi model Sugeno

Dengan pertimbangan bahwa model Sugeno ini lebih cocok untuk sistem nonlinier daripada model Mamdani.

***Fuzzy* C-means (FCM)**

Fuzzy clustering adalah salah satu teknik untuk menentukan cluster optimal dalam suatu ruang vektor yang didasarkan pada bentuk normal Euklidian untuk jarak antar vektor. FCM adalah salah satu teknik pengklusteran data yang mana keberadaan tiap-tiap titik data dalam suatu cluster ditentukan oleh derajad keanggotaanya. Teknik ini pertama kali diperkenalkan oleh Jim Bezdek tahun 1981. *Fuzzy clustering* sangat handal terutama bagi pemodelan *fuzzy* untuk identifikasi aturan-aturannya.

Konsep dasar FCM adalah menentukan pusat cluster, yang akan menandai lokasi rata-rata untuk tiap-tiap cluster. Tiap-tiap titik data sekitar rata-rata memiliki derajad keanggotaan cluster. Dengan cara memperbaiki pusat cluster dan derajad keanggotaan tiap-tiap titik data secara berulang maka pusat cluster akan bergerak menuju pusat cluster yang tepat. Pengulangan ini didasarkan pada minimisasi fungsi objektif, antara lain menggunakan metode *recursive least square* (RLS). Informasi keluaran dari FCM dapat digunakan untuk membangun *fuzzy inference system*

Penggabungan Sistem Fuzzy

Sistem *fuzzy* telah menunjukkan kecakapannya dalam menyelesaikan berbagai persoalan klasifikasi, modeling dan kontrol. Hingga tahun 1990 basis pengetahuan sistem *fuzzy* didapat dari pemindahan pengetahuan pakar kedalam sistem inferensinya. Tahun itu kemampuan belajar mulai ditambahkan pada sistem *fuzzy*. Dua pendekatan yang berhasil untuk penambahan kecakapan belajar bagi sistem ini adalah gabungan sistem *fuzzy* dengan *neural* dan *evolutionary*. Gabungan sistem itu dikenal dengan *neuro-fuzzy* dan *genetic fuzzy systems (GFSs).* Perkembangan sistem hybrid terus berlanjut misalnya menjadi *genetic fuzzy clustering* atau *genetic neuro-fuzzy systems.* Tipe yang mengemuka dari GFS adalah *genetic fuzzy rule-base systems* (GFRBSs), yaitu proses pembelajaran bagi system *fuzzy* berbasis algoritma genetik. Pembelajaran secara teknik diartikan sebagai usaha optimisasi dan adaptasi berbagai parameter system yang berbeda sehingga tercapai system yang efisien dan efektif dalam proses pencariannya.

Pada awal 1990 Zadeh mengusulkan konsep komputasi lunak (*soft computing*), yang dengan cepat meluas kepada penerapan industri dalam sistem *aerospace*, sistem komunikasi, sistem tenaga listrik, otomasi pabrikasi, robotika, elektronika daya dan transportasi. Teknik-teknik komputasi lunak memberikan kontribusi terhadap solusi-solusi dalam daerah aplikasi dengan data yang tidak tepat (*imprecise*) dan pengetahuan tidak lengkap (*incomplete knowledge*) seperti diagnostik, sistim identifikasi, estimasi dan kontrol.

Berikut penerapan sukses GFS pada permasalahan dunia nyata seperti kontrol, pabrikasi, produk konsumsi, transportasi, pemodelan dan pengambilan keputusan. Bonissone, menggambarkan skema optimisasi pengontrol *fuzzy* yang mengatur kecepatan kereta barang. Tujuan pengaturan kecepatan supaya kereta bergerak dengan nyaman pada lintasan dan tekanan pada penggandeng antar gerbong tetap halus dan paling kecil. Penalaan menggunakan AG terhadap parameter pengontrol dilakukan bertahap, pertama menyesuaikan faktor skala variabel masukan dan keluaran, yaitu variabel yang paling berpengaruh terhadap prilaku pengontrol secara menyeluruh. Tahap kedua, AG menala fungsi keanggotaan masing-masing variabel. Pengontrol *fuzzy* ini kemudian dicobakan secara simulasi pada lintasan yang berbeda. Secara umum penalaan pengontrol *fuzzy* meningkatkan ketepatan penjejakan lintasan dan kehalusan tekanan pada penggandeng antar gerbong. Dalam (Cordon O,2004) Bonissone menyimpulkan, bahwa sumbangan terbesar dari peningkatan kemampuan pengontrol *fuzzy* oleh penalaan

faktor skala, terutama meminimalkan biaya komputasi, sedangkan sumbangan oleh penalaan fungsi keanggotaan hanya kecil saja terhadap peningkatan kinerja pengontrol.

Hwang (Cordon O,2004), menyelesaikan persoalan yang serupa, transportasi. Persoalan yang diangkat olehnya adalah optimisasi waktu perjalanan (*trip-time*) dan konsumsi energi kereta api cepat. *Fuzzy* c-means clustering dan AG, mengidentifikasi hubungan antara struktur dan parameter model linguistik dengan kecepatan sebagai masukan dan keluaran sebagai *trip time* dan konsumsi energi. Parameter model diidentifikasi oleh AG yang digabungkan (*hybrid*) dengan teknik *hill-climbing.* Metode ini diterapkan untuk strategi kontrol perencanaan rail kereta api cepat di Korea. Secara ekonomis, dilaporkan, kereta beroperasi dengan *trip-time margin* kurang dari 7% dan penghematan energi 5%.

Voget, menyajikan skema optimisasi *multi-objective*, di mana suatu pengontrol *fuzzy* mengatur prosedur seleksi dan fungsi kebugaran AG. Pendekatan ini disebut algoritma evolusioner *fuzzy*, yaitu sistim *fuzzy* mengatur sumber daya dan parameter-parameter AG seperti laju mutasi, ukuran populasi dan tekanan selektif untuk memperbaiki kinerja. Pada pendekatan tertentu, aturan-aturan *fuzzy* secara heuristik mengizinkan AG untuk mengidentifikasi himpunan solusi Pareto-optimal. Yang yang didasarkan pada bias (*deviasi*) penyimpangan antara populasi yang ada dan fungsi *multi-objective*, pengontrol *fuzzy* memutuskan skema seleksi dan fungsi kebugaran yang akan dipakai AG untuk mencapai sasaran optimisasi sekaligus meliputi Pareto-optimal. Pendekatan sudah diberlakukan bagi optimisasi jadwal jaringan kereta api, dengan sasaran untuk mengurangi waktu tunggu penumpang ketika pergantian kereta, dan pada waktu yang sama mengurangi biaya investasi baru untuk memperbaiki infrastruktur. Hasil dari optimisasi AG adalah suatu kurva *cost-benefit* yang menunjukkan pengaruh dari investasi terhadap waktu tunggu penumpang, yang menjelaskan *trade-off* antara keduanya.

Mizutani, mengusulkan suatu sistim gabungan *fuzzy-neuro-genetic* untuk memprediksi warna, menggunakan komputer, suatu tantangan dalam produksi cat. Arsitektur pabrikasi pencampuran warna cat, tidak bisa dikarakterisasi oleh GFS konvensional dimana algoritma optimisasi terhadap basis pengetahuan *fuzzy*nya ditentukan oleh pakar, sebagai gantinya, pengetahuan mewarnai pakar ahli dinyatakan oleh aturan-aturan *fuzzy-neuro* berbasis AG.

Daftar aplikasi GFS diatas memperlihatkan kontribusi pengontrol *fuzzy* berbasis AG, pada pemecahan permasalahan industri dan komersial. Manfaat utama dari optimisasi pada industri dan komersial adalah solusi atas biaya-rendah dan kualitas tinggi.

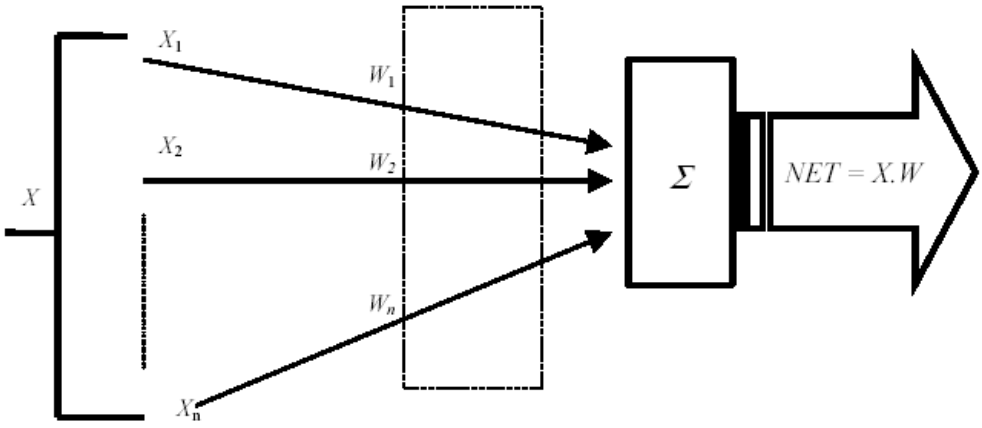
BAB IV
Dasar Teori Jaringan Syaraf Tiruan

Secara prinsip, JST ini dibangkitkan oleh serangkaian masukan yang masing-masing menggambarkan keluaran neuron yang lain. Setiap masukan dikalikan dengan suatu faktor pembobot tertentu dan kemudian semua masukan terbobot itu dijumlahkan untuk menentukan tingkat aktivitas suatu neuron, seperti terlihat pada Gambar 1. Perhitungan sekelompok neuron dalam jaringan yang menghasilkan keluaran *NET* merupakan perkalian matriks sederhana, artinya

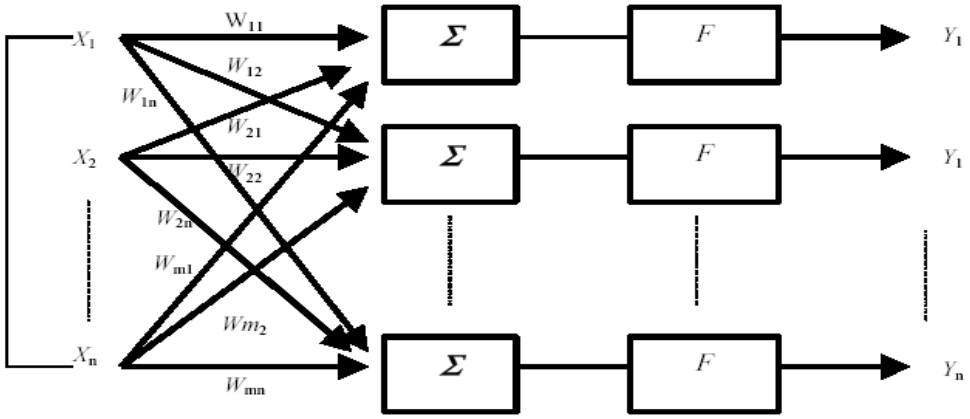
$$NET = X.W$$

di mana *NET* dan *X* merupakan vektor baris.

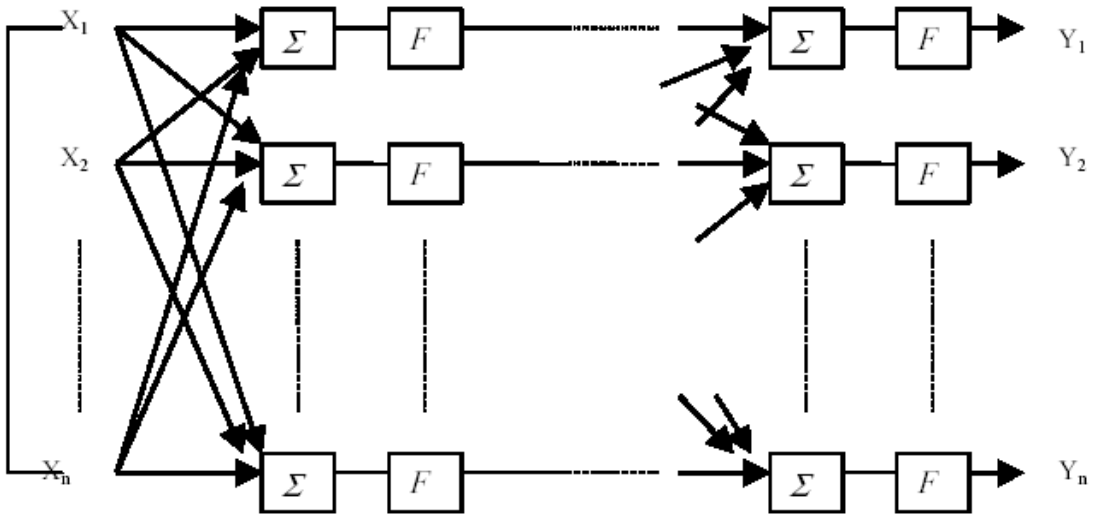
Secara sederhana setiap neuron menghasilkan keluaran dari jumlah masukan terbobot pada jaringan. Pembobot dinyatakan berupa elemen-elemen matriks *W* dengan dimensi matriks adalah *m* baris dan *n* kolom, di mana *m* merupakan jumlah masukan dan *n* menunjukkan jumlah neuron. Misalnya, pembobot dengan yang menghubungkan masukan keempat ke neuron ketiga dinyatakan dengan ***W43***. Dengan cara ini, penghitungan sekelompok neuron *NET* yang menghasilkan keluaran ***N*** untuk sebuah lapisan merupakan perkalian matriks sederhana. Artinya, ***N = XW***, di mana ***N*** dan ***X*** merupakan vektor baris. Konsep dasar *single layer* di atas mengilhami para pakar untuk membangun JST *multi layer* yang ditunjukkan Gambar 1.



Gambar 1. Jaringan Syaraf Tiruan.



Gambar 2. Jaringan Syaraf Tiruan Lapis Tunggal.



Gambar 3. Jaringan Syaraf Tiruan Lapis Banyak.

Model JST yang telah digambarkan di atas adalah model yang *non-recurrent*, yaitu model tanpa umpan balik dari keluaran ke masukan, sedangkan sistem biologis menunjukkan sifat umpan balik. Oleh karena itu, untuk menyempurnakan sistem JST dibuat umpan balik terhadap bobot yang akan mempengaruhi lapisan pertama dan seterusnya sehingga dalam proses belajar selalu diberikan harga bobot yang teriterasi sampai mencapai suatu keadaan yang optimal [2] .

JST umpan maju yang mempunyai lebih dari satu lapisan tersembunyi memerlukan pemetaan matematis yang teratur dan rapi supaya aliran proses terlihat

dengan jelas dan baik. Notasi *i* menunjukkan urutan masukan dari suatu elemen aktif dan *j* adalah urutan elemen proses itu sendiri. Elemen aktif pada JST terdiri dari elemen-elemen tersembunyi. Dengan demikian, secara matematis proses umpan maju dalam sistem jaringan ini adalah :

$$v=\sum w.x \qquad (1)$$

dimana :

x = masukan

w = bobot

v = keluaran

Keluaran dari elemen proses di atas merupakan fungsi transfer yang umumnya menggunakan fungsi sigmoid dengan persamaan umum :

$$y = \frac{1}{1+\exp(-v)} \qquad (2)$$

$$\delta_j^{(l)} = e_j^{(L)} \cdot o_j \cdot \left(1 - o_j\right)$$

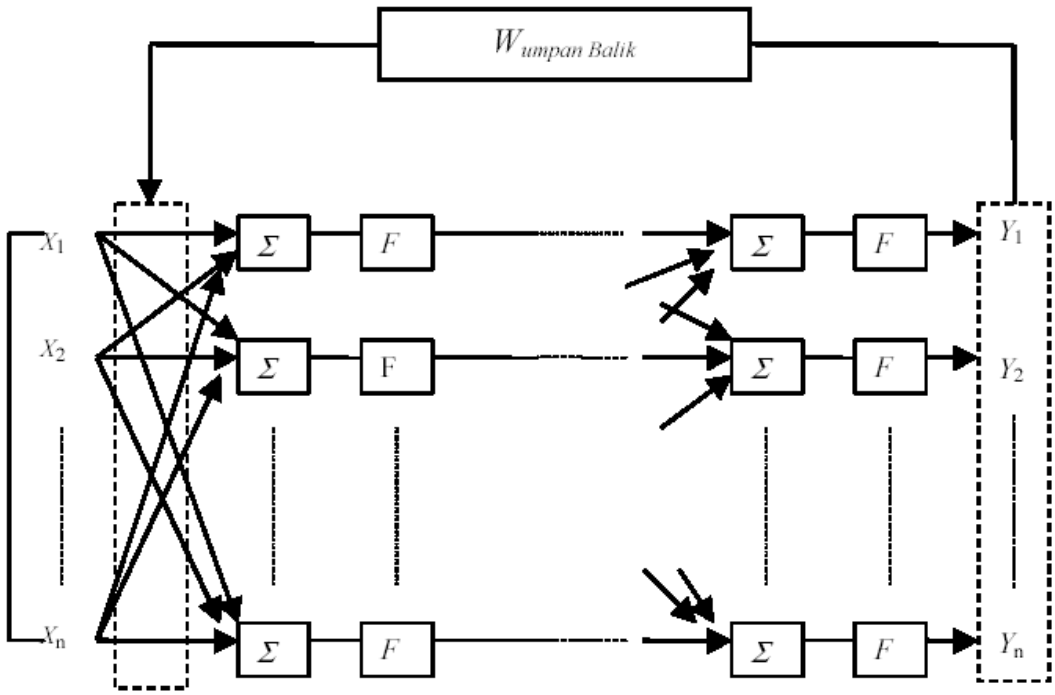
Harga error dari suatu JST merupakan hasil pengurangan keluaran JST terhadap keluaran JST yang kita inginkan. Dengan demikian dapat kita tulis sebagai berikut

$$e_j(n) = d_j(n)-y_j(n) \qquad (3)$$

di mana *dj*(n) adalah elemen keluaran yang kita inginkan.

Error di atas adalah error untuk masing-masing parameter keluaran, sehingga perhitungan error total :

$$d_k = \frac{1}{2}\sum_{j=1}^{N} e_f^2 \qquad (4)$$



Gambar 4. JST Multi Layer dengan Umpan Balik.

Pengaturan konfigurasi jaringan akan melibatkan pertimbangan-pertimbangan berdasarkan error total yang dihasilkan jaringan, sehingga prediksi optimalisasi terhadap perubahan-perubahan konfigurasi yang akan diuji lebih optimistik dan terarah.

Proses belajar sistem JST ini menggunakan pelatihan perambatan balik, di mana proses yang terjadi kebalikan dari proses umpan maju. Perambatan balik dimulai dari error keluaran jaringan yang diumpankan ke belakang merambat sampai ke lapisan aktif terdepan. Perambatan error ke belakang ini berfungsi sebagai parameter pembanding dalam proses perbaikan nilai bobot yang telah diperoleh. Persamaan gradien lokal pada masing-masing lapisan jaringan dapat ditulis sebagai berikut :

$$\delta_j^{(l)} = y_j^{(l)} \cdot \left(1 - y_j^{(l)}\right) \cdot \sum_k \left(\delta_k^{(l+1)} \cdot w_{kj}^{(l+1)}\right)$$

di mana *j* menunjukkan posisi neuron pada lapisan tersembunyi *l*. Untuk lapisan keluaran di mana $l = L$, persamaan yang digunakan :di mana *j* adalah posisi neuron pada lapisan keluaran. Dengan demikian, perbaikan bobot yang akan dilakukan oleh JST tergantung pada hasil perhitungan gradien lokal yang melibatkan perambatan error dari lapisan

keluaran sampai lapisan aktif terdepan. Aliran proses generalisasi jaringan pada lapisan tersembunyi *l* adalah sebagai berikut 6) :

$$w_{ji}^{(l)}(n+1) \quad = \quad w_{ji}^{(l)}(n) + \alpha\left(w_{ji}^{(l)}(n) - w_{ji}^{(l)}(n-1)\right) + \eta\delta_j^{(l)}(n) \cdot y_i^{(l-1)}(n)$$

di mana :

n = nilai laju belajar

α = konstanta momentum.

BAB V
KONSEP DASAR ALGORITMA GENETIKA

Algoritma Genetika pada dasarnya adalah program komputer yang mensimulasikan proses evolusi. Dalam hal ini populasi dari kromosom dihasilkan secara random dan memungkinkan untuk berkembang biak sesuai dengan hukum-hukum evolusi dengan harapan akan menghasilkan individu kromosom yang prima. Kromosom ini pada kenyataannya adalah kandidat penyelesaian dari masalah, sehingga bila kromosom yang baik berkembang, solusi yang baik terhadap masalah diharapkan akan dihasilkan.

Algoritma Genetika ini banyak dipakai pada aplikasi bisnis, teknik maupun pada bidang keilmuan. Algoritma ini dapat dipakai untuk mendapatkan solusi yang tepat untuk masalah optimal dari satu variabel atau multi variabel. Sebelum algoritma ini dijalankan, masalah apa yang ingin dioptimalkan itu harus dinyatakan dalam fungsi tujuan, yang dikenal dengan fungsi *fitness*. Jika nilai *fitness* semakin besar, maka sistem yang dihasilkan semakin baik. Walaupun pada awalnya semua nilai *fitness* kemungkinan sangat kecil (karena algoritma ini menghasilkannya secara random), sebagian akan lebih tinggi dari yang lain. Kromosom dengan nilai *fitness* yang tinggi ini akan memberikan probabilitas yang tinggi untuk bereproduksi pada generasi selanjutnya. Sehingga untuk setiap generasi pada proses evolusi, fungsi *fitness* yang mensimulasikan seleksi alam, akan menekan populasi kearah *fitness* yang meningkat.

Algoritma genetika sangat tepat digunakan untuk penyelesaian masalah optimasi yang kompleks dan sukar diselesaikan dengan menggunakan metode yang konvensional. Sebagaimana halnya proses evolusi di alam, suatu algoritma genetika yang sederhana umumnya terdiri dari tiga operator yaitu: ***operator reproduksi***, ***operator crossover*** (persilangan) dan ***operator mutasi***. Struktur umum dari suatu algoritma genetika dapat didefinisikan dengan langkah-langkah sebagai berikut:

1. Membangkitkan populasi awal, Populasi awal ini dibangkitkan secara random sehingga didapatkan solusi awal. Populasi itu sendiri terdiri dari sejumlah kromosom yang merepresentasikan solusi yang diinginkan.
2. Membentuk generasi baru, Dalam membentuk digunakan tiga operator yang telah disebut di atas yaitu operator reproduksi/seleksi, crossover dan mutasi. Proses ini dilakukan berulang-ulang sehingga didapatkan jumlah kromosom yang cukup untuk

membentuk generasi baru dimana generasi baru ini merupakan representasi dari solusi baru.

3. Evaluasi solusi, Proses ini akan mengevaluasi setiap populasi dengan menghitung nilai *fitness* setiap kromosom dan mengevaluasinya sampai terpenuhi kriteria berhenti. Bila kriteria berhenti belum terpenuhi maka akan dibentuk lagi generasi baru dengan mengulangi langkah 2. Beberapa kriteria berhenti yang sering digunakan antara lain:

- Berhenti pada generasi tertentu.
- Berhenti setelah dalam beberapa generasi berturut-turut didapatkan nilai *fitness* tertinggi tidak berubah.
- Berhenti bila dalam *n* generasi berikut tidak didapatkan nilai *fitness* yang lebih tinggi.

Pada uraian berikut, membahas aplikasi algoritma genetika pada bidang sistem distribusi air bersih. Algoritma genetika adalah suatu algoritma pencarian yang bersifat stokastik, berbasis pada mekanisme dari seleksi alam dan genetika. Algoritma genetika sangat tepat digunakan untuk penyelesaian masalah optimasi yang kompleks dan sukar diselesaikan dengan menggunakan metode yang konvensional.

Suatu algoritma genetika yang sederhana umumnya terdiri dari tiga operator [1] yaitu operator reproduksi, operator crossover dan operator mutasi. Struktur umum dari suatu algoritma genetika dapat didefinisikan dengan langkah-langkah sebagai berikut:

1. Membangkitkan populasi awal Populasi awal ini dibangkitkan secara random sehingga didapatkan solusi awal. Populasi itu sendiri terdiri dari sejumlah kromosom yang merepresentasikan solusi yang diinginkan.
2. Membentuk generasi baru Dalam membentuk digunakan tiga operator yang telah disebut di atas yaitu operator reproduksi/seleksi, crossover dan mutasi. Proses ini dilakukan berulang-ulang sehingga didapatkan jumlah kromosom yang cukup untuk membentuk generasi baru dimana generasi baru ini merupakan representasi dari solusi baru.
3. Evaluasi solusi Proses ini akan mengevaluasi setiap populasi dengan menghitung nilai fitness setiap kromosom dan mengevaluasinya sampai terpenuhi criteria berhenti. Bila kriteria berhenti belum terpenuhi maka akan dibentuk lagi generasi baru dengan mengulangi langkah 2.

Beberapa kriteria berhenti yang sering digunakan antara lain:

- Berhenti pada generasi tertentu.
- Berhenti setelah dalam beberapa generasi berturut-turut didapatkan nilai fitness tertinggi tidak berubah.
- Berhenti bila dalam *n* generasi berikut tidak didapatkan nilai fitness yang lebih tinggi. Sebelum algoritma genetika dilakukan, ada dua hal penting yang harus dilakukan yaitu pendefinisian kromosom dan fungsi fitness atau fungsi obyektif. Dua hal ini juga memegang peranan penting dalam algoritma genetika untuk menyelesaikan suatu masalah

A. Implementasi pada Mikrokontroler

Algoritma genetika dapat diimplementasikan pada mikrokontroler 89C51 yang merupakan salah satu keluarga mikrokontroler MCS51. Beberapa prosedur standar telah dikembangkan untuk menunjang implementasi algoritma genetika pada mikrokontoler khususnya mikrokontroler MCS51. Prosedur tersebut adalah:

- Prosedur *Random_8* dan *Random_16* Prosedur ini disediakan untuk membangkitkan bilangan random. Jenis bilangan random yang dihasilkan adalah *pseudo random.* Prosedur random_8 berfungsi untuk membangkitkan bilangan random 8 bit dan prosedur random_16 untuk membangkitkan bilangan random 16 bit.
- Prosedur *Create_1st_Generation* Prosedur ini disediakan untuk membangkitkan populasi awal.
- Prosedur *Crossover* Prosedur ini disediakan melakukan proses crossover dari dua kromosom dan menghasilkan dua kromosom baru. Jenis crossover yang digunakan adalah one-cut point crossover.
- Prosedur *Mutation* Prosedur ini disediakan untuk melakukan proses mutasi. Jenis mutasi yang digunakan adalah mutasi bit.
- Prosedur *Roulette_ Wheel_Selection* Prosedur ini disediakan untuk melakukan proses seleksi dengan menggunakan metode roulette wheel.
- Prosedur *Elitsm_Selection* Prosedur ini disediakan untuk melakukan proses seleksi dengan menggunakan metode elitsm dimana satu kromosom terbaik akan terpilih. Beberapa spesifikasi yang perlu diperhatikan adalah jumlah kromosom per populasi maksimum 5 kromosom, jenis kromosom adalah binary bit string dan panjang maksimumnya 16 bit

B. Representasi Kromosom

Jenis kromosom yang digunakan adalah binary bit string dengan jumlah bit bergantung pada:

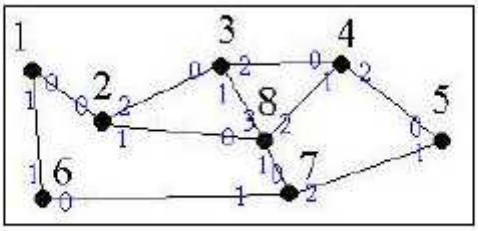
- Jumlah titik.
- Jumlah percabangan terbanyak untuk satu titik.

Panjang kromosom (dalam bit) dapat dihitung dengan menggunakan persamaan berikut:

$$P = C \, x \, (T\text{-}1) \quad (1)$$

Dimana *C* adalah jumlah bit binary dari jumlah percabangan terbanyak dan *T* adalah jumlah titik. Pada gambar 1 ada 8 titik dan jumlah percabangan terbanyak adalah 4 cabang (membutuhkan 2 bit binary) maka dengan menggunakan persamaan 1 didapatkan panjang kromosomnya 14 bit.

Setiap kromosom mengandung informasi rute dari satu titik ke titik yang lain. Bit-bit dalam kromosom menunjukkan nomor cabang pada titik-titik yang ada. Bila nomor cabang tidak ada maka dilakukan perulangan. Untuk lebih jelas lihat contoh berikut yang menggunakan peta seperti pada gambar 1.



Gambar 1. Contoh peta jalan yang diimplementasikan

Start awal = titik 1

Kromosom = 01 00 11 10 00 11 10

Rute = 1 – 6 – 7 – 8 – 4 – 3 – 2 – 3

C. Fungsi Fitness

Dalam sistem ini, permasalahan optimasi adalah mencari rute terpendek. Karena itu fungsi fitness yang digunakan dapat didefinisikan sebagai jarak total dari rute yang ada. Fungsi fitness tersebut dapat direpresentasikan oleh persamaan berikut:

$$F = \sum_{Rute} Jarak \; antar \; titik$$

D. Operator Algoritma Genetika

Dalam aplikasi ini metode seleksi yang digunakan adalah roulette wheel dan elitsm. Dalam seleksi roulette wheel, peluang terpilihnya suatu kromosom sebanding dengan besar nilai fitness kromosom tersebut. Karena permasalahan optimasi mencari rute terpendek (minimum), maka pada seleksi roulette wheel, nilai fitness masing-masing kromosom diubah dengan menggunakan persamaan berikut:

$F_n' = F_{max} + F_{min} - F_n$

Dalam aplikasi ini, jenis crossover yang digunakan adalah one-cut point crossover sedangkan jenis mutasi yang mutasi bit.

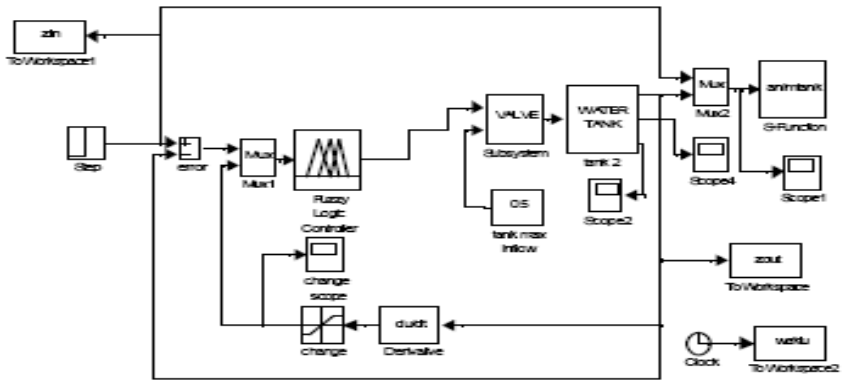
PENGKODEAN ALGORITMA GENETIKA

Algoritma Genetika adalah suatu algoritma pencarian yang meniru mekanisme dari genetika alam. Algoritma Genetika ini banyak dipakai pada aplikasi bisnis, teknik maupun pada bidang keilmuan.Algoritma ini dapat dipakai untuk mendapatkan solusi yang tepat untuk masalah optimal dari satu variabel atau multi variabel. Sebelum Algoritma ini dijalankan, masalah apa yang ingin dioptimalkan itu harus dinyatakan dalam fungsi tujuan, yang dikenal dengan fungsi fitness. Jika nilai fitness semakin besar, maka sistem yang dihasilkan semakin baik. Operasi yang dilakukan adalah reproduksi, crossover, dan mutasi untuk mendapatkan sebuah solusi menurut nilai fitnessnya. Selanjutnya konstruksi dasar dari Algoritma Genetika adalah
sebagai berikut:

- ✓ Pendefinisian Chromosome
- ✓ Pendefinisian Fungsi Fitness
- ✓ Membangkitkan Sebuah Populasi Awal
- ✓ Reproduksi
- ✓ Crossover
- ✓ Mutasi

A. Desain Kendali Logika Fuzzy Dengan Algoritma Genetika

Pada bagian ini Algoritma Genetika akan dipakaimendapatkan optimasi dari kendali logika fuzzyuntuk suatu model simulasi yaitu pada water level control. Gambar simulasi water level control tampak di berikut ini:



B. Algoritma Genetika dengan Pengkodean Solusi sebagai Bilangan Biner

Urutan proses dari algoritma genetika dengan pengkodean solusi sebagai bilangan biner diperlihatkan dengan menggunakan contoh sederhana oleh Gen, et.al. (1997) dan Kosasih, et.al. (2005). Berikut diuraikan secara singkat kelima tahapan proses yang dilakukan.

B.1. Kodifikasi Solusi dan Pembentukan Generasi Awal

Kodifikasi solusi sebagai bilangan biner pada algoritma genetika merupakan cara yang paling umum digunakan. Untuk contoh *kasus* yang sedang dikerjakan, karena nilai *ITP* sebagai solusi yang dicari adalah dua desimal, maka kode bilangan biner yang digunakan harus dapat mewakili rentang nilai antara 267 ÷ 4541 (13 *bit* = 8191). Nilai desimal 8191 mewakili nilai *ITP* = 45,41 cm dan nilai desimal 0 mewakili nilai *ITP* = 2.67 cm. Nilai *ITP* antara dihitung secara proporsional.

Setiap *bit* merupakan *gen* yang membentuk individu. Dengan demikian, masing-masing individu memiliki 13 *gen* sebagai solusi. Misalkan, salah satu individu yang terbentuk secara random adalah 0010111101001 (= 1513). Nilai desimal 1513 ini dikonversikan ke dalam nilai *ITP* secara proporsional menjadi:

$$ITP = 2.67 + \frac{1513}{8191} * (45.41 - 2.67) = 10.57 \text{ cm}$$

Generasi awal dibentuk dengan jumlah individu (populasi) yang biasanya ditetapkan antara 10 ÷ 100, dengan jumlah *gen* per individu terdiri dari bilangan random sebesar 13 *bit*. Secara umum, makin besar populasi yang dibentuk, maka akan makin besar pula kemungkinan solusi optimum yang dapat dihasilkan dari algoritma genetika. Akan tetapi, sebagai konsekwensi, waktu pemrosesankomputer akan menjadi lebih lama.

B.2. Proses Pertukaran *Gen*

Pertukaran *gen* antara dua individu yang dinyatakan sebagai kode bilangan biner dapat dilakukan dengan menggunakan operator genetika konvensional. Kedua individu yang mengalami proses pertukaran *gen* ditentukan secara random. Hasil yang diperoleh adalah dua individu baru sebagai turunannya.

Cara yang mudah untuk melakukan pertukaran *gen* adalah berdasarkan titik potong yang juga ditentukan secara random. Potongan *gen* sebelah kiri titik potong dari satu

individu induk digabungkan dengan potongan *gen* sebelah kanan titik potong dari individu induk lainnya.

Tabel 1: Contoh proses pertukaran *gen*

2 individu yang mengalami proses pertukaran gen		
kode bilangan biner	nilai desimal	nilai ITP
0010111101001	1513	10.56
1011101111010	6010	34.03

→

2 individu turunan yang terbentuk		
kode bilangan biner	nilai desimal	nilai ITP
0010101111010	1402	9.99
1011111101001	6121	34.61

Proses pertukaran *gen* untuk dua individu induk yang terpilih diilustrasikan pada Tabel 2, dimana titik potong dimisalkan terjadi pada *gen* ke lima. Seperti terlihat, dua individu turunan yang dihasilkan memiliki nilai *ITP* yang berbeda. Selanjutnya, jumlah individu yang mengalami proses pertukaran *gen* pada satu generasi ditentukan secara random berdasarkan tingkat probabilitas pertukaran *gen* yang diijinkan. ika tingkat probabilitas pertukaran *gen* yang diijinkan adalah 25%, maka untuk generasi dengan populasi yang terdiri dari 10 individu, jumlah individu yang akan mengalami proses pertukaran *gen* kurang lebih adalah 2 atau 3.

B.3. Proses Perubahan *Gen*

Perubahan *gen* merupakan operator genetika kedua dan hanya bekerja pada beberapa *gen* dari individu yang melakukan penyesuaian diri terhadap kondisi lingkungan sekitar. Proses perubahan *gen* terjadi agar makhluk hidup dapat terus bertahan hidup dengan kwalitas yang lebih baik.

Pada algoritma genetika, proses perubahan *gen* yang menghasilkan *gen* yang lebih baik dapat membuat individu tetap bertahan dalam proses seleksi dan diharapkan akan dapat makin mendekati solusi optimum. Sebaliknya, proses perubahan *gen* yang menghasilkan *gen* yang lebih buruk dapat membuat individu tersebut tereliminasi dalam proses seleksi.

Tabel 3: Contoh proses perubahan *gen*

2 individu yang mengalami proses perubahan gen		
kode bilangan biner	nilai desimal	nilai ITP
1011111101001	6121	34.61

→

2 individu turunan yang terbentuk		
kode bilangan biner	nilai desimal	nilai ITP
0011111101001	2025	13.24

Untuk individu yang dinyatakan dengan kode bilangan biner, proses perubahan *gen* juga dilakukan dengan menggunakan operator genetika konvensional, yaitu mengubah nilai 0 menjadi 1, atau sebaliknya (lihat Tabel 2). Jumlah *gen* yang mengalami proses perubahan*gen* pada satu generasi ditentukan secara random berdasarkan tingkat

probabilitas perubahan *gen* yang diijinkan. Untuk contoh kasus pencarian nilai *ITP* dengan populasi misalnya sebesar 10x13 *gen*, jika tingkat probabilitas yang diijinkan adalah 10%, maka jumlah *gen* yang akan mengalami proses perubahan *gen* pada satu generasi kurang lebih 13.

B.4. Proses Evaluasi

Proses evaluasi pada algoritma genetika dimaksudkan untuk menghitung nilai kecocokan terhadap solusi optimum dari setiap individu turunan yang telah mengalami proses pertukaran *gen* dan perubahan *gen*. Untuk contoh kasus pencarian nilai *ITP*, nilai kecocokan yang dimaksud adalah nilai *deviasi* yang dihitung dari *Pers (2)* dengan mensubstitusikan nilai *ITP* yang mewakili setiap individu turunan tersebut. Dalam hal ini, individu terbaik adalah individu yang memiliki nilai *deviasi* positip terkecil. Individu turunan yang dihasilkan tidak harus selalu lebih baik dari pada individu induk.

B.5. Proses Seleksi

Proses seleksi dilakukan untuk memilih individu induk dan individu turunan berdasarkan nilai kecocokan yang diperoleh di atas untuk membentuk generasi baru yang lebih baik ke arah solusi optimum yang dicari. Ada tiga ketentuan dasar yang harus dipertimbangkan dalam melakukan proses seleksi, yaitu:

a. Jumlah populasi pada setiap generasi baru harus selalu dipertahankan tetap (*n*).

b. Fungsi seleksi yang cocok harus dipilih sesuai dengan jenis aplikasinya. Empat alternatif fungsi seleksi yang disediakan pada program Genetika (Kosasih, 2005), termasuk:

1. Fungsi seleksi random; generasi baru dipilih secara random berdasarkan kurva distribusi kumulatif nilai kecocokan, baik yang berasal dari individu induk, maupun dari individu turunan dengan prosedur, sbb.:

i). hitung probabilitas nilai kecocokan,

$$P_i = (\sum_{j=1}^{i} deviasi_j) / (\sum_{j=1}^{2n} deviasi_j)$$

ii). ambil nilai random, P_{random} *(= 0 ÷ 1)*

iii). pilih individu *i* yang memenuhi kondisi ($P_i < P_{random} \leq P_{i+1}$)

iv). ulangi langkah ii) dan iii) untuk *n* individu yang membentuk generasi baru

2. Fungsi seleksi random terkoreksi; sama seperti alternatif fungsi 1 di atas tetapi individu terbaik dari individu induk atau individu turunan harus selalu disertakan dalam generasi baru.
3. Fungsi seleksi turunan terkoreksi; semua individu turunan dipilih sebagai generasi baru, kecuali individu terburuk yang digantikan oleh individu terbaik dari generasi induk; dan untuk menghindari duplikasi, individu pengganti harus lebih baik dibandingkan dengan yang terbaik dari individu turunan.
4. Fungsi seleksi unggul; semua individu yang memiliki nilai kecocokan terbaik dipilih dari gabungan antara individu induk dan individu turunan sebagai generasi baru.

c. Duplikasi individu pada generasi baru harus dapat dicegah untuk menghindari proses pencarian terperangkap pada solusi optimum lokal. Disamping itu, nilai fungsi individu yang saling berdekatan seharusnya juga tidak disukai karena akan mempersempit ruang pencarian.

Sejauh ini, satu siklus regenerasi telah diuraikan secara singkat. Individu terbaik dari generasi baru yang dihasilkan mungkin masih belum merupakan solusi optimum yang dicari. Proses regenerasi kemudian terus berlanjut sampai solusi optimum dapat diperoleh sesuai dengan ketentuan tambahan yang telah dijelaskan di atas.

C. Algoritma Genetika dengan Pengkodean Solusi sebagai Bilangan Riil

Algoritma genetika dengan pengkodean solusi sebagai bilangan riil telah banyak diaplikasikan. Kekhususan dari kode bilangan riil adalah dalam hal operator genetika yang digunakan. Operator non-konvensional, baik untuk pertukaran *gen*, maupun untuk perubahan *gen*, harus dipilih secara tepat sesuai dengan aplikasi yang dikerjakan.

Kelima tahapan proses dari algoritma genetika untuk solusi yang dinyatakan sebagai bilangan riil diuraikan secara rinci berikut ini. Satu siklus regenerasi mulai dari generasi awal sampai pada generasi kedua dijelaskan dengan menggunakan contoh yang lengkap.

C.1. Kodifikasi Solusi dan Pembentukan Generasi Awal

Kodifikasi solusi sebagai bilangan riil pada dasarnya dapat dilakukan secara langsung berdasarkan rentang nilai solusi yang diijinkan. Sebagai contoh, 10 individu yang

mungkin terbentuk secara random untuk menjadi generasi awal, beserta dengan nilai *deviasi* masing-masing yang dihitung dari *Pers (2)*, adalah sebagai berikut:

Tabel 4: Contoh generasi awal yang terbentuk

No Individu	1	2	3	4	5	6	7	8	9	10
ITP (cm)	24.16	37.56	12.62	41.17	24.88	22.15	39.02	18.87	29.96	4.13
deviasi	2.1705	3.8163	-0.0083	4.1663	2.2776	1.8566	3.9615	1.2906	2.9641	-2.8714

Dari ketentuan yang telah ditetapkan di atas, individu 8 merupakan solusi yang terbaik dan individu 4 merupakan solusi yang terburuk dari generasi awal ini.

C.2. Proses Pertukaran *Gen*

Proses pertukaran *gen* antara dua individu induk yang dinyatakan sebagai kode bilangan riil dapat dilakukan dengan menggunakan operator genetika non-konvensional. Ada dua jenis operator genetika non-konvensional yang dikenal, yaitu operator genetika aritmatik dan operator genetika direksional (Gen, et.al., 1997).

Operator genetika aritmatik pada dasarnya tidak selalu dapat memberikan kepastian bahwa individu turunan yang dihasilkan menjadi lebih baik dibandingkan dengan individu induknya. Untuk contoh kasus pencarian nilai *ITP*, ketentuan tentang solusi optimum dapat disertakan ke dalam operator genetika direksional, sehingga individu turunan yang dihasilkan diharapkan akan selalu lebih baik.

Kedua individu yang mengalami proses pertukaran *gen* tetap perlu ditentukan secara random. Akan tetapi, hasil yang diperoleh dalam hal ini hanyalah satu individu baru saja sebagai turunannya, seperti terlihat pada *Pers (4)* khusus untuk nilai *deviasi* yang positip. Untuk nilai *deviasi* yang negatip atau yang berbeda tanda, *Pers (4)* dapat disesuaikan seperlunya. Setelah proses pertukaran *gen* dilakukan, nilai *ITP'* seharusnya akan lebih baik dari pada, atau setidaknya sama dengan, nilai ITP_1 atau nilai ITP_2

$ITP' = ITP_1 + R * (ITP_1 - ITP_2)$: *untuk* $ITP_1 < ITP_2$ *dan nilai deviasi > 0.0*

Atau

$ITP' = ITP_2 + R * (ITP_2 - ITP_1)$: *untuk* $ITP_1 > ITP_2$ *dan nilai deviasi > 0.0*

dimana: *ITP'* = individu turunan

ITP_1, ITP_2 = dua individu induk yang mengalami proses pertukaran *gen*

R = bilangan random *(= 0 ÷ 1)*

Misalkan, untuk tingkat probabilitas pertukaran *gen* sebesar 25%, individu 1 dan individu 7 terpilih secara random untuk mengalami proses pertukaran *gen* pada generasi awal ini.

Nilai *R* pada contoh ini adalah 0.70. Individu turunan yang dihasilkan berdasarkan *Pers (4)* adalah 13.76.

Kesepuluh individu setelah mengalami proses pertukaran *gen* diperlihatkan pada Tabel 5. Terlihat bahwa individu 7 menguat dengan nilai *deviasi* turun dari 3.9615 (lihat Tabel 4) menjadi 0.2512; dan, individu 7 juga lebih baik dari individu 1 (sebagai individu induk kedua) yang memiliki nilai *deviasi* 2.1705.

Tabel 5: Contoh proses pertukaran *gen*

No Individu	1	2	3	4	5	6	7	8	9	10
ITP (cm)	24.16	37.56	12.62	41.17	24.88	22.15	13.76	18.87	29.96	4.13
deviasi	2.1705	3.8163	-0.0083	4.1663	2.2776	1.8566	0.2512	1.2906	2.9641	-2.8714

C.3. Proses Perubahan *Gen*

Proses perubahan *gen* pada individu turunan yang dinyatakan sebagai kode bilangan riil juga dilakukan dengan menggunakan operator genetika non-konvensional. Seperti halnya pada proses pertukaran *gen*, untuk contoh kasus pencarian nilai *ITP*, operator genetika direksional lebih cocok digunakan, seperti terlihat pada *Pers (5)*. Pertukaran *gen* ini dilakukan relatif terhadap nilai rata-rata dari semua individu yang ada.

$$ITP' = ITP_1 - R * deviasi_1 * |gradient|$$

dimana: *ITP' = individu turunan*

ITP_1 = individu induk yang mengalami proses perubahan gen

R = bilangan random (= 0 ÷ 1)

$$gradient = \frac{\frac{\sum_{i=1}^{jml\ populasi,\ n} ITP_i}{n} - ITP_1}{\frac{\sum_{i=1}^{jml\ populasi,\ n} deviasi_i}{n} - deviasi_1}$$

Pers (5) merupakan rumus interpolasi dinamis, yang serupa dengan *Pers (3)*. Sebagai model *stochastic*, *Pers (5)* memiliki variabel tambahan *R* yang harus ditentukan secara random.

Pada generasi awal yang sedang dianalisis, untuk tingkat probabilitas 10%, individu 1, 2, 4 dan 10 terpilih secara random untuk mengalami proses pertukaran *gen*. Pada contoh ini, nilai *R* adalah 0.62.

Keempat individu turunan yang dihasilkan beserta dengan individu-individu lainnya diperlihatkan pada Tabel 6. Setelah mengalami proses pertukaran *gen*, keempat individu menjadi lebih baik dengan nilai *deviasi* yang mendekati nol (bandingkan dengan Tabel 5).

Tabel 6: Contoh proses perubahan *gen*

No Individu	1	2	3	4	5	6	7	8	9	10
ITP (cm)	21.29	22.00	12.62	22.87	24.88	22.15	13.76	18.87	29.96	11.63
deviasi	1.7154	1.8318	-0.0083	1.9714	2.2776	1.8566	0.2512	1.2906	2.9641	-0.2432

C.4. Proses Evaluasi

Proses evaluasi untuk solusi yang menggunakan kode bilangan riil pada dasarnya tidak berbeda dengan yang menggunakan kode bilangan biner (lihat Butir III.4). Sebagai contoh, nilai *deviasi* sebagai nilai kecocokan yang dihitung berdasarkan *Pers (2)* telah turut disertakan pada Tabel 6, dan beberapa tabel sebelumnya.

C.5. Proses Seleksi

Proses seleksi untuk solusi yang menggunakan kode bilangan riil juga tidak berbeda dengan yang menggunakan kode bilangan biner (lihat Butir III.5). Sebagai contoh, untuk fungsi seleksi unggul, generasi kedua dapat dihasilkan dengan menseleksi individu yang terbaik dari individu induk dan individu turunan. Hasil yang diperoleh diperlihatkan pada Tabel 7, dimana individu 1 ÷ 9 dipilih dari individu turunan (lihat Tabel 6) dan hanya individu 10 dipilih dari individu induk (lihat Tabel 4).

Tabel 7: Contoh generasi kedua yang dihasilkan

No Individu	1	2	3	4	5	6	7	8	9	10
ITP (cm)	21.29	22.00	12.62	22.87	24.88	22.15	13.76	18.87	11.63	24.16
deviasi	1.7154	1.8318	-0.0083	1.9714	2.2776	1.8566	0.2512	1.2906	-0.2432	2.1705

REPRESENTASI KROMOSOM ALGORITMA GENETIKA DALAM BENTUK BINER

A. Pendahuluan

Sebuah nilai solusi yang dihasilkan dalam algoritma genetika disebut dengan *chromosome* [1]. *Chromosome* tersusun atas gen-gen yang merupakan representasi nilai variabel dalam suatu fungsi yang ingin dicari solusinya. Chromosome dapat dibentuk dari bilangan numerik, biner ataupun karakter. Representasi chromosome dalam bentuk biner adalah bentuk yang banyak digunakan karena mempermudah pemrograman didalam proses algoritma genetika seperti crossover dan mutasi. Untuk memudahkan dalam mengingat selanjutnya *chromosome* yang tersusun atas gen-gen dalam bentuk biner dapat kita sebut sebagai *chromosome* biner.

Sebelum diproses dalam algoritma genetika, solusi yang dibangkitkan baik dalam bentuk integer atau float harus dirubah dulu ke dalam bentuk biner. Demikian pula sebaliknya setelah proses-proses dalam algoritma genetika selesai maka nilai chromosome biner tersebut harus dirubah kembali ke bentuk semula yaitu berupa bilangan integer atau float.

Berikut adalah contoh proses membentuk chromosome biner dari suatu bilangan integer. Misalkan diketahui fungsi $f(x)=x3+2x+1$, algoritma genetika digunakan untuk mendapatkan nilai x yang optimal dalam range solusi $9 \leq x \leq 30$. Maka chromosome biner x dapat dibentuk dengan bilangan biner yang merepresentasikan nilai integer antara 9 sampai 30. Langkah pertama adalah menentukan panjang bilangan biner yang dapat menjangkau nilai integer mulai dari 9 sampai 30 atau dapat juga ditulis $2n>30$ dimana n = panjang biner. Yang ingin kita tentukan adalah nilai n yang jika 2 dipangkatkan dengan n nilainya mendekati 30. Jika n = 4, maka 24=16, nilai integer yang dapat direpresentasikan hanya mulai dari 0 sampai 15. Apabila n = 6 maka integer yang dapat direpresentasikan adalah dari 0 sampai 63, tetapi hal ini tidak efektif dalam penggunakaan memori karena yang dibutuhkan hanya untuk mengkodekan desimal dari 0-30. Sehingga nilai n yang paling sesuai adalah 5 karena 25=32, range nilai integer yang dapat direpresentasikannya adalah dari 0 sampai 31.

Setelah diketahui panjang biner proses selanjutnya adalah mengubah bilangan integer menjadi biner. Algoritma untuk mengubah bilangan integer ke biner sangat sederhana yaitu dengan mendapatkan nilai modulus (sisa hasil pembagian) dari pembagian bilangan dengan 2 sebanyak panjang biner (n kali).

Sebagai contoh adalah merubah bilangan integer 23 ke dalam bentuk biner dengan panjang biner $n = 5$:

Langkah-langkah:

1. Lakukan proses pembagian bilangan dengan 2 sebanyak 5 kali iterasi, dan dapatkan nilai modulusnya:

    23 dibagi 2 sama dengan 11 sisa 1

    11 dibagi 2 sama dengan 5 sisa 1

    5 dibagi 2 sama dengan 2 sisa 1

    2 dibagi 2 sama dengan 1 sisa 0

    1 dibagi 2 sama dengan 0 sisa 1

2. Dapatkan nilai binernya dengan cara menyimpan nilai modulus dimulai dari bawah atau dari pembagian yang terakhir.

Maka biner untuk 23 adalah: 10111

B. Pengkodean Bilangan Floating

Prinsip untuk merubah bilangan float ke biner adalah dengan merubah bilangan float tersebut menjadi bilangan integer terlebih dahulu, setelah itu bilangan integer tersebut baru dirubah dalam biner. Untuk mengubah dari bilangan float ke integer adalah dengan mengalikan bilangan tersebut dengan 10^m dimana m = jumlah angka dibelakang koma dari bilangan float yang akan dirubah ke biner. Jika ruang solusi dari algoritma genetika adalah antara $a \leq x \leq b$ dimana x adalah bilangan float, maka [2]:

$$2^{n-1} < (b-a) * 10^m \leq 2^n - 1$$

Demikian pula proses sebaliknya untuk merubah dari bilangan biner ke bentuk bilangan float maka proses pertama kali adalah dengan mendapatkan nilai integer dari bilangan biner tersebut baru kemudian dihitung nilai floatnya.

Rumus yang dapat digunakan untuk merubah bilangan biner ke float adalah:

$$x = a + integer * \frac{(b-a)}{2^n - 1}$$

dimana *integer* adalah nilai integer dari bilangan biner yang akan dirubah.

Sebagai contoh range solusi adalah -3.2<*x*<12,4 dengan nilai *x* yang diinginkan adalah bilangan

float dengan kepresisian 4 angka dibelakang koma. Untuk merepresentasikan dalam bentuk chromosome biner langkahnya adalah sebagai berikut:

Pertama adalah menentukan panjang biner yang diperlukan menggunakan persamaan (1):

$$2^{n-1} < (b-a) * 10^m \leq 2^n - 1$$

karena jumlah angka dibelakang koma yang diinginkan adalah 4 digit maka *m* = 4, sehingga:

$$= 2^{n-1} < (12{,}4 - (-3{,}2)) * 10^4 \leq 2^n - 1$$

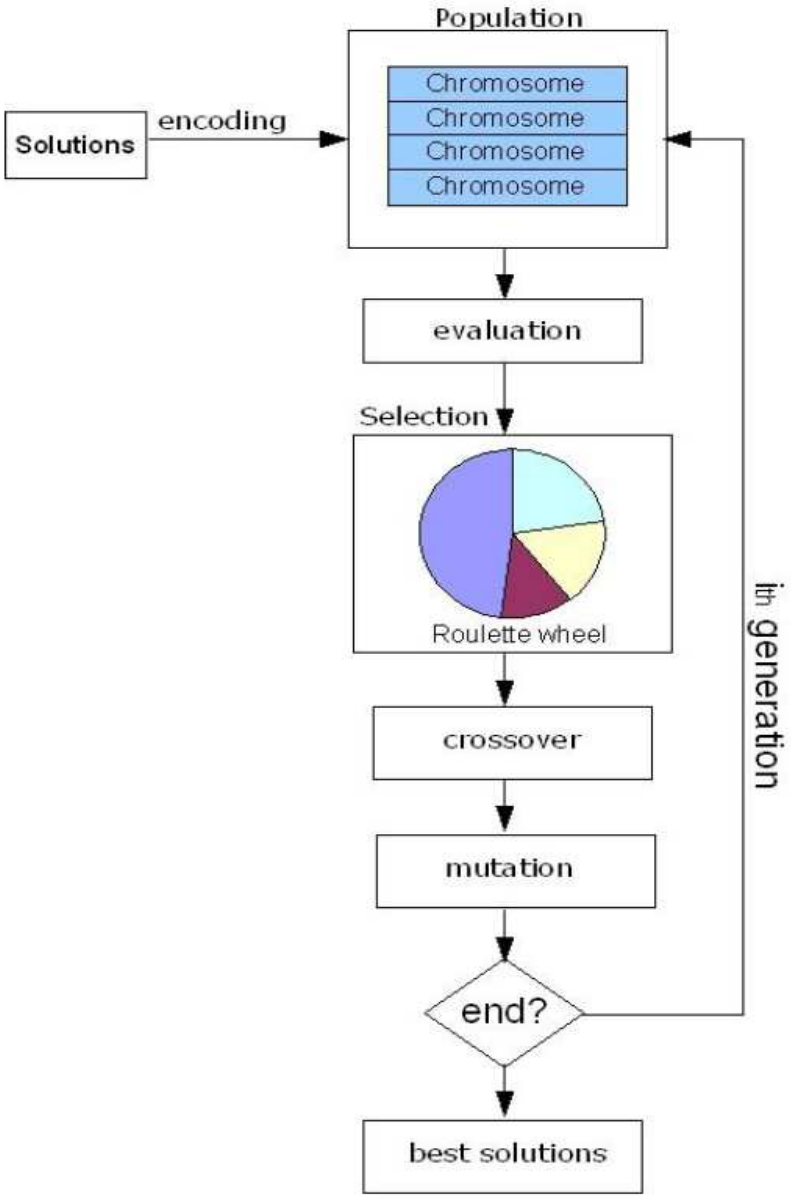
$$= 2^{n-1} < (15{,}6 * 10^4) \leq 2^n - 1$$

$$= 2^{n-1} < 156000 \leq 2^n - 1$$

Nilai *n* yang memenuhi persamaan diatas adalah *n* = 18 karena $2^{17} < 156000 < 2^{18}$

INTERPRETASI OPERATOR ALGORITMA GENETIKA

GA adalah algoritma yang diinspirasi teori evolusi. Flowchart GA adalah sebagai berikut:



Misalkan kita mempunyai persamaan a+2b+3c+4d = 30, kita ingin mencari nilai a, b, c, dan d yang memenuhi persamaan diatas. Disini kita menggunakan GA untuk menyelesaikan permasalahan diatas.

Yang dicari adalah variabel a, b, c, dan d yang memenuhi persamaan diatas, maka objektif dari GA adalah meminimalkan f(x) = Min|((a + 2b + 3c + 4d) - 30)|

Karena ada 4 variabel dalam persamaan yaitu a, b, c, dan d maka kita dapat melakukan pengkodean seperti dibawah ini:

Batasan nilai-nilai variabel a, b, c, dan d adalah bilangan integer 0 sampai 30.

A. Inisialisasi

Misalkan kita gunakan jumlah populasi dalam satu generasi adalah 6, maka:

Chromosome[1] = [a;b;c;d] = [12;05;23;08]

Chromosome[2] = [a;b;c;d] = [02;21;18;03]

Chromosome[3] = [a;b;c;d] = [10;04;13;14]

Chromosome[4] = [a;b;c;d] = [20;01;10;06]

Chromosome[5] = [a;b;c;d] = [01;04;13;19]

Chromosome[6] = [a;b;c;d] = [20;05;17;01]

B. Evaluasi Chromosome

Kita hitung fungsi objektif dari chromosome yang telah dibangkitkan:

Fitness[1] = Abs((12 + 2*05 + 3*23 + 4*08) - 30)
= Abs((12 + 10 + 69 + 32) - 30)
= Abs(123 - 30)
= 93
Fitness[2] = Abs((02 + 2*21 + 3*18 + 4*03) - 30)
= Abs((02 + 42 + 54 + 12) - 30)
= Abs(110 - 30)
= 80
Fitness[3] = Abs((10 + 2*04 + 3*13 + 4*14) - 30)
= Abs((10 + 08 + 39 + 56) - 30)
= Abs(113 - 30)
= 83
Fitness[4] = Abs((20 + 2*01 + 3*10 + 4*06) - 30)
= Abs((20 + 02 + 30 + 24) - 30)
= Abs(76 - 30)
= 46
Fitness[5] = Abs((01 + 2*04 + 3*13 + 4*19) - 30)
= Abs((01 + 08 + 39 + 76) - 30)
= Abs(124 - 30)
= 94
Fitness[6] = Abs((20 + 2*05 + 3*17 + 4*01) - 30)
= Abs((20 + 10 + 51 + 04) - 30)
= Abs(85 - 30)
= 55

C. Seleksi Chromosome

1. Karena diinginkan chromosome yang mempunyai fitness yang lebih kecil mempunyai probabilitas untuk terpilih kembali lebih besar maka digunakan inverse.

Q[1] = 1 / **Fitness**[1]

= 1 / 93
= 0.0108

Q[2] = 1 / **Fitness**[2]
= 1 / 80
= 0.0125

Q[3] = 1 / **Fitness**[3]
= 1 / 83
= 0.0120

Q[4] = 1 / **Fitness**[4]
= 1 / 46
= 0.0217

Q[5] = 1 / **Fitness**[5]
= 1 / 94
= 0.0106

Q[6] = 1 / **Fitness**[6]
= 1 / 55
= 0.0182

Total = 0.0108 + 0.0125 + 0.0120 + 0.0217 + 0.0106 + 0.0182
= 0.0859

Rumus untuk mencari probabilitas: **P**[i] = **Q**[i] / **Total**

P[1] = 0.0108 / 0.0859
= 0.1252

P[2] = 0.0125 / 0.0859
= 0.1456

P[3] = 0.0120 / 0.0859
= 0.1403

P[4] = 0.0217 / 0.0859
= 0.2532

P[5] = 0.0106 / 0.0859
= 0.1239

P[6] = 0.0182 / 0.0859
= 0.2118

Dari probabilitas diatas dapat kita lihat kalau **Chromosome** ke 4 yang mempunyai fitness paling kecil mempunyai probabilitas untuk terpilih pada generasi selanjutnya lebih besar dari **Chromosome** lainnya.

Untuk proses seleksi kita gunakan roulete wheel, untuk itu kita harus mencari dahulu nilai cumulative probabilitasnya:

C[1] = 0.1252

C[2] = 0.1252 + 0.1456
= 0.2708

C[3] = 0.1252 + 0.1456 + 0.1403
= 0.4111

C[4] = 0.1252 + 0.1456 + 0.1403 + 0.2532
= 0.6643

C[5] = 0.1252 + 0.1456 + 0.1403 + 0.2532 + 0.1239
= 0.7882

C[6] = 0.1252 + 0.1456 + 0.1403 + 0.2532 + 0.1239 + 0.2118
= 1.0

Setelah dihitung cumulative probabilitasnya maka proses seleksi menggunakan roulete-wheel dapat dilakukan. Prosesnya adalah dengan membangkitkan bilangan acak **R** dalam range 0-1.

Jika **R**[k] < **P**[1] maka pilih chromosome 1 sebagai induk, selain itu pilih chromosome ke-k sebagai induk dengan syarat **P**[k-1] < **R** < **P**[k]. Kita putar roulete wheel sebanyak jumlah populasi yaitu 6 kali bangkitkan bilangan acak **R**) dan pada tiap putaran, kita pilih satu chromosome untuk populasi baru.

Misal:

R[1] = 0.201

R[2] = 0.284

R[3] = 0.099

R[4] = 0.822

R[5] = 0.398

R[6] = 0.501

Angka acak pertama **R**[1] adalah lebih besar dari **P**[1] dan lebih kecil daripada **P**[2] maka pilih Chromosome[2] sebagai chromosome pada populasi baru, dari bilangan acak yang telah dibangkitkan diatas maka populasi barunya adalah:

NewChromosome[1] = **Chromosome**[2]

NewChromosome[2] = **Chromosome**[3]

NewChromosome[3] = **Chromosome**[1]

NewChromosome[4] = **Chromosome**[6]

NewChromosome[5] = **Chromosome**[3]

NewChromosome[6] = **Chromosome**[4]

Dengan demikian Chromosome dalam populasi menjadi:

Chromosome[1] = [02;21;18;03]

Chromosome[2] = [10;04;13;14]

Chromosome[3] = [12;05;23;08]

Chromosome[4] = [20;05;17;01]

Chromosome[5] = [10;04;13;14]

Chromosome[6] = [20;01;10;06]

D. Crossover

Metode yang digunakan salah satunya adalah one-cut point, yaitu memilih secara acak satu posisi dalam chromosome induk kemudian saling menukar sub-chromosome. Chromosome yang dijadikan induk dipilih secara acak dan jumlah Chromosome yang mengalami crossover dipengaruhi oleh parameter *crossover probability* (**ρc**) yang telah ditentukan.

Pseudo-code untuk proses crossover adalah sebagai berikut:

```
begin
    k← 0;
    while(k≤populasi) do
    R[k] ← random(0-1);
    if (R[k] < ρc ) then
    select Chromosome[k] as parent;
    end;
    k = k + 1;
    end;
end;
```

Misal kita tentukan crossover probability adalah sebesar 25%, maka diharapkan dalam satu generasi ada 50% Chromosome (3 chromosome) dari satu generasi mengalami proses crossover. Prosesnya adalah sebagai berikut:

Pertama kita bangkitkan bilangan acak **R** sebanyak jumlah populasi

R[1] = 0.191
R[2] = 0.259
R[3] = 0.760
R[4] = 0.006
R[5] = 0.159
R[6] = 0.340

Maka Chromosome ke k akan dipilih sebagai induk jika **R**[k] < **ρc**, dari bilangan acak **R** diatas maka yang dijadikan induk adalah **Chromosome**[1], **Chromosome**[4] dan **Chromosome**[5].

Setelah melakukan pemilihan induk proses selanjutnya adalah menentukan posisi crossover. Ini dilakukan dengan cara membangkitkan bilangan acak dengan batasan 1 sampai (panjang **Chromosome**-1), dalam kasus ini bilangan acak yang dibangkitkan adalah 1 – 3. Misalkan didapatkan posisi crossover adalah 1 maka **Chromosome** induk akan dipotong pada **Sub-Chromosome** ke 1

kemudian potongan **Sub-Chromosome** tersebut saling ditukarkan antar induk.

Chromosome[1] >< **Chromosome**[4]

Chromosome[4] >< **Chromosome**[5]

Chromosome[5] >< **Chromosome**[1]

Posisi *cut-point* crossover dipilih menggunakan bilangan acak 1-3 sebanyak jumlah crossover yang terjadi, misal

C[1] = 1

C[2] = 1

C[3] = 2

Chromosome[1] = **Chromosome**[1] >< **Chromosome**[4]

= [02;21;18;03] >< [20;05;17;01]

= [02;05;17;01]

Chromosome[4] = **Chromosome**[4] >< **Chromosome**[5]

= [20;05;17;01] >< [10;04;13;14]

= [20;04;13;14]

Chromosome[5] = **Chromosome**[5] >< **Chromosome**[1]

= [10;04;13;14] >< [02;21;18;03]

= [10;04;18;03]

Dengan demikian populasi Chromosome setelah mengalami proses crossover menjadi:

Chromosome[1] = [02;05;17;01]

Chromosome[2] = [10;04;13;14]

Chromosome[3] = [12;05;23;08]

Chromosome[4] = [20;04;13;14]

Chromosome[5] = [10;04;18;03]

Chromosome[6] = [20;01;10;06]

E. Mutasi

Jumlah chromosome yang mengalami mutasi dalam satu populasi ditentukan oleh parameter mutation rate. Proses mutasi dilakukan dengan cara mengganti satu sub-chromosome yang dipilih secara acak dengan suatu nilai baru. Prosesnya adalah sebagai berikut. Pertama kita hitung dahulu panjang total **Sub-Chromosome** yang ada dalam satu populasi. Dalam kasus ini panjang total **Sub-Chromosome** adalah **Total-Sub-Chromosome** = jumlah **Sub-Chromosome** * jumlah populasi

= 4 * 6

= 24

Untuk memilih posisi **Sub-Chromosome** yang mengalami mutasi dilakukan dengan cara membangkitkan bilangan integer acak antara 1 sampai **Total-Sub-Chromosome** yaitu 1 sampai 24. Jika bilangan acak yang kita bangkitkan lebih kecil daripada variabel *mutation rate* (**ρm**) maka pilih posisi tersebut sebagai sub-chromosome yang mengalami mutasi. Misal **ρm** kita tentukan 10% maka diharapkan 10% (0.1) dari **Total-Sub-Chromosome** dalam populasi yaitu sebanyak:

jumlah mutasi = 0.1 * 24
= 2.4
= 2

Misalkan setelah kita bangkitkan bilangan acak terpilih posisi **Sub-Chromosome** 12 dan 18 yang mengalami mutasi. Dengan demikian yang akan mengalami mutasi adalah **Chromosome** ke-3 **Sub- Chromosome** nomor 4 dan **Chromosome** ke-5 **Sub-Chromosome** nomor 2. Maka nilai **Sub-Chromosome** pada posisi tersebut kita ganti dengan bilangan acak 0-30 (sesuai batasan nilai **Chromosome**).

Misalkan bilangan acak yang terbangkitkan adalah 2 dan 5. Maka populasi chromosome setelah
mengalami proses mutasi adalah:

Chromosome[1] = [02;05;17;01]

Chromosome[2] = [10;04;13;14]

Chromosome[3] = [12;05;23;**02**]

Chromosome[4] = [20;04;13;14]

Chromosome[5] = [10;**05**;18;03]

Chromosome[6] = [20;01;10;06]

Setelah proses mutasi maka kita telah melakukan satu iterasi dalam GA atau disebut dengan satu generasi. Maka fungsi fitness setelah satu generasi adalah:

Chromosome[1] = [02;05;17;01]

Fitness[1] = Abs((02 + 2*05 + 3*17 + 4*01) - 30)
= Abs((2 + 10 + 51 + 4) - 30)
= Abs(67 - 30)
= 37

Chromosome[2] = [10;04;13;14]

Fitness[2] = Abs((10 + 2*04 + 3*13 + 4*14) - 30)
= Abs((10 + 8 + 33 + 56) - 30)
= Abs(107 - 30)
= 77

Chromosome[3] = [12;05;23;02]

Fitness[3] = Abs((12 + 2*05 + 3*23 + 4*02) - 30)

= Abs((12 + 10 + 69 + 8) - 30)
= Abs(87 - 30)
= 47

Chromosome[4] = [20;04;13;14]
Fitness[4] = Abs((20 + 2*04 + 3*13 + 4*14) - 30)
= Abs((20 + 8 + 39 + 56) - 30)
= Abs(123 - 30)
= 93

Chromosome[5] = [10;05;18;03]
Fitness[5] = Abs((10 + 2*05 + 3*18 + 4*03) - 30)
= Abs((10 + 10 + 54 + 12) - 30)
= Abs(86 - 30)
= 56

Chromosome[6] = [20;01;10;06]
Fitness[6] = Abs((20 + 2*01 + 3*10 + 4*06) - 30)
= Abs((20 + 2 + 30 + 24) - 30)
= Abs(76 - 30)
= 46

PROSES DAN ANALISIS ALGORITMA GENETIKA UNTUK NILAI MAXIMUM

A. Contoh Tahapan Proses Dari Algoritma Genetika

Berikut diuraikan secara rinci contoh lima tahapan proses utama dari algoritma genetika, sesuai dengan yang telah diilustrasikan pada Gambar 1.

A.1. Kodifikasi Solusi dan Pembentukan Generasi Awal

Proses kodifikasi dari solusi yang dicari ke dalam *chromosome* merupakan isu kunci dalam algoritma genetika. Untuk fungsi sinus yang dianalisis, solusi yang dicari adalah nilai *x1* dan *x2*. Jika ketelitian nilai *x1* dan *x2* adalah empat desimal, maka kode *binary* yang digunakan harus dapat mewakili rentang nilai -30,000 ÷ 120,000 (18 *bits* = 262,143) untuk nilai *x1*, dan rentang nilai 41,000 ÷ 58,000 (15 *bits* = 32767) untuk nilai *x2*. Setiap *bit* merupakan *gen* yang membentuk *chromosome*. Dengan demikian, masingmasing *chromosome* memiliki 33 *gen* yang merupakan gabungan dari nilai *x1* dan *x2* sebagai solusi.

Generasi awal kemudian dibentuk dengan jumlah *chromosome* yang biasanya ditetapkan antara 10 ÷ 100, dengan jumlah *gen* per *chromosome* terdiri dari bilangan random sebesar 33 *bits*. Makin besar jumlah populasi yang dibentuk, maka akan makin besar pula kemungkinan solusi optimum yang dapat dihasilkan dari algoritma genetika. Tetapi, sebagai konsekwensi, waktu pemrosesan komputer akan menjadi lebih lama. Sebagai contoh, dalam proses pencarian nilai fungsi sinus maksimum, 10 *chromosome* yang mungkin terbentuk secara random untuk menjadi generasi awal adalah sbb.:

C_1 = 0 0 0 1 0 1 1 0 0 1 0 1 0 0 0 1 1 0 1 0 0 1 0 1 0 0 1 1 1 0 1 0 1
= { 22854 ; 19061 } = 19.483

C_2 = 1 0 1 0 0 1 1 1 1 0 0 0 1 0 1 1 0 0 1 1 0 0 0 1 1 0 0 0 1 1 1 1 0
= { 171564 ; 25374 } = 19.300

C_3 = 0 1 1 0 0 0 1 1 1 0 1 1 0 0 0 1 0 1 1 1 1 1 0 1 0 1 1 0 1 0 0 1 1
= { 102085 ; 31443 } = 23.913

C_4 = 1 1 1 1 0 1 1 1 0 1 0 1 0 0 0 1 0 1 1 0 0 0 0 1 1 1 0 1 0 1 0 1 1
= { 253253 ; 17323 } = 31.458

C_5 = 1 1 1 1 0 0 0 0 0 0 1 1 1 1 0 1 1 1 0 0 0 0 1 1 0 1 1 1 0 0 1 0 1
= { 246007 ; 1765 } = 28.774

C_6 = 1 1 0 1 0 1 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 0 0 1
= { 217348 ; 505 } = 27.949

C_7 = 0 0 0 0 0 1 0 1 1 0 0 1 0 1 0 1 1 0 0 0 0 0 0 1 0 1 1 0 1 1 1 1 0
= { 5718 ; 734 } = 26.561

C_8 = 1 1 0 0 1 0 1 1 0 0 0 0 1 0 0 0 1 1 1 0 1 1 1 0 0 1 1 1 1 1 1 1 0
= { 207907 ; 23806 } = 23.120

C_9 **= 0 1 1 0 1 0 1 0 0 1 1 1 1 0 0 1 0 1 1 0 1 1 1 1 0 1 1 1 1 1 0 1 1**
= { 109029 ; 24315 } = 16.736
C_{10} = 0 0 1 0 0 0 0 1 1 0 1 1 0 1 0 0 0 1 0 1 1 1 1 1 1 0 1 0 1 0 0 0 0
= { 34513 ; 16208 } = 24.327

Nilai *x1* dan *x2* sebagai solusi dan nilai fungsi sinus untuk masing-masing *chromosome* juga telah dihitung; dimana, *chromosome* 4 merupakan solusi yang terkuat {*f* (*x1* , *x2*) = 31.458} dan *chromosome* 9 merupakan solusi yang terlemah {*f* (*x1* , *x2*) = 16.736}.

A.2. Proses Crossover

Crossover dan juga *mutation* merupakan dua operator genetika utama. Secara umum, operator genetika dapat diklasifikasikan ke dalam 3 kelas, yaitu:

- Operator konvensional
- Operator aritmatika
- Operator direksional

Untuk *chromosome* dengan kode *binary*, proses *crossover* dilakukan dengan menggunakan operator konvensional, seperti yang akan diuraikan berikut ini. Dua operator *crossover* lainnya dijelaskan oleh Gen, et.al. (2005). Proses *crossover* bekerja pada dua *chromosome* melalui pertukaran *gen* (kawin silang) untuk menghasilkan dua *chromosome* baru sebagai turunannya. Cara yang mudah untuk dapat melakukan pertukaran *gen* adalah dengan menentukan titik potong secara random. Potongan *gen* sebelah kiri titik potong dari satu *chromosome* induk kemudian digabungkan dengan potongan *gen* sebelah kanan titik potong dari *chromosome* induk lainnya (lihat Gambar 1).

Jumlah *chromosome* yang mengalami proses *crossover* pada satu generasi ditentukan secara random berdasarkan tingkat probabilitas *crossover* yang diijinkan. Pada tingkat probabilitas *crossover* yang cukup tinggi, proses pencarian solusi optimum dapat menjelajah ke ruang explorasi yang lebih luas sehingga kemungkinan terperangkap pada nilai optimum lokal yang salah dapat dihindari. Akan tetapi, jika tingkat probabilitas *crossover* yang diijinkan terlalu tinggi, maka waktu pemrosesan komputer dapat menjadi lebih lama, dan proses pencarian solusi optimum menjadi *redundant*.

Kesepuluh *chromosome* setelah mengalami proses *crossover* diperlihatkan pada daftar berikut. Untuk tingkat probabilitas *crossover* sebesar 25%, dua *chromosome* 5 dan 9 ditetapkan secara random untuk mengalami proses *crossover* pada generasi ini. Proses *crossover* dilakukan pada *gen* ketujuh yang juga ditetapkan secara random. Terlihat bahwa

chromosome 5 melemah dengan nilai fungsi turun dari 28.774 menjadi 11.060. Sedangkan, *chromosome* 9 menguat dengan nilai fungsi naik dari 16.736 menjadi 22.393.

C_1' = 0 0 0 1 0 1 1 0 0 1 0 1 0 0 0 1 1 0 1 0 0 1 0 1 0 0 1 1 1 0 1 0 1
= { 22854 ; 19061 } = 19.483

C_2' = 1 0 1 0 0 1 1 1 1 0 0 0 1 0 1 1 0 0 1 1 0 0 0 1 1 0 0 0 1 1 1 1 0
= {171564 ; 25374 } = 19.300

C_3' = 0 1 1 0 0 0 1 1 1 0 1 1 0 0 0 1 0 1 1 1 1 1 0 1 0 1 1 0 1 0 0 1 1
= {102085 ; 31443 } = 23.913

C_4' = 1 1 1 1 0 1 1 1 0 1 0 1 0 0 0 1 0 1 1 0 0 0 0 1 1 1 0 1 0 1 0 1 1
= {253253 ; 17323 } = 31.458

$\boldsymbol{C_5}$' = 1 1 1 1 0 0 **1 0 0 1 1 1 1 0 0 1 0 1 1 0 1 1 1 1 0 1 1 1 1 1 0 1 1**
= **{ 248293** ; **24315 }** = **11.060**

$C_{6'}$ = 1 1 0 1 0 1 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 1 1 1 1 1 0 0 1
= {217348 ; 505 } = 27.949

C_7' = 0 0 0 0 0 1 0 1 1 0 0 1 0 1 0 1 1 0 0 0 0 0 0 1 0 1 1 0 1 1 1 1 0
= { 5718 ; 734 } = 26.561

C_8' = 1 1 0 0 1 0 1 1 0 0 0 0 1 0 0 0 1 1 1 0 1 1 1 0 0 1 1 1 1 1 1 1 0
= {207907 ; 23806 } = 23.120

$\boldsymbol{C_9}$' = 0 1 1 0 1 0 **0 0 0 0 1 1 1 1 0 1 1 1 0 0 0 0 1 1 0 1 1 1 0 0 1 0 1**
= **{ 106743** ; **1765 }** = **22.393**

C_{10}' = 0 0 1 0 0 0 0 1 1 0 1 1 0 1 0 0 0 1 0 1 1 1 1 1 1 0 1 0 1 0 0 0 0
= { 34513 ; 16208 } = 24.327

III.3. Proses *Mutation*

Mutation merupakan operator genetika kedua dan hanya bekerja pada beberapa *gen* yang melakukan penyesuaian diri terhadap kondisi lingkungan sekitar. Proses *mutation* terjadi agar makhluk hidup dapat terus bertahan hidup dengan kwalitas yang lebih baik. Pada algoritma genetika, proses *mutation* yang menghasilkan *gen* yang lebih baik dapat membuat *chromosome* tetap bertahan dalam proses seleksi dan diharapkan akan dapat makin mendekati solusi optimum. Sebaliknya, proses *mutation* yang menghasilkan *gen* yang lebih buruk dapat membuat *chromosome* tereliminasi dalam proses seleksi.

Untuk *chromosome* dengan kode *binary*, proses *mutation* juga dilakukan dengan menggunakan operator konvensional, yaitu mengubah nilai 0 menjadi 1, atau sebaliknya (lihat Gambar 1). Jumlah *gen* yang mengalami proses *mutation* pada satu generasi ditentukan secara random berdasarkan tingkat probabilitas *mutation* yang diijinkan.

Seperti halnya pada proses *crossover*, jika tingkat probabilitas *mutation* yang diijinkan terlalu rendah, maka ada *gen* yang mungkin seharusnya berguna tidak pernah ikut terseleksi. Sebaliknya, jika tingkat probabilitas *mutation* terlalu tinggi, maka *chromosome* turunan mungkin akan mulai kehilangan kesamaan dengan *chromosome* induknya

sehingga membuat algoritma genetika kehilangakemampuan untuk belajar dari sejarah dalam proses pencarian solusi optimum.

Untuk contoh pencarian nilai fungsi sinus maksimum, populasi yang ada terdiri dari 10x33 *gen*. Jika tingkat probabilitas *mutation* yang diijinkan adalah 10%, maka jumlah *gen* yang akan mengalami proses *mutation* pada satu generasi adalah kurang lebih 33.
Pada daftar berikut ini diperlihatkan kesepuluh *chromosome* setelah mengalami proses *mutation*.

C_1 = 0 0 0 1 0 1 **0** 0 0 1 0 1 0 0 0 1 1 0 **0** 0 **1** 1 0 1 0 0 1 1 1 0 1 **1** 1
= { 20806 ; 6775 } = 19.995

C_2" = 1 0 1 0 0 1 1 1 1 0 0 0 1 0 1 **0** 0 0 1 1 0 0 0 1 1 0 0 0 1 1 1 1 0
= { 171560 ; 25374 } = 19.298

C_3" = 0 1 1 0 0 0 1 1 **0** **1** 1 1 0 0 0 1 0 1 1 1 **0** 1 0 **0** 0 1 1 0 1 0 0 1 1
= { 101829 ; 26835 } = 16.081

C_4" = 1 1 1 1 0 1 1 1 0 1 **1** 1 0 0 0 1 0 1 1 0 0 0 0 1 **0** 1 0 1 0 1 0 1 1
= { 253381 ; 17067 } = 28.350

C_5" = 1 1 1 1 0 0 1 0 0 1 1 1 1 0 0 1 0 1 1 0 1 1 1 1 0 1 1 **0** 1 1 **1** 1 1
= **{ 248293** ; **24287 }** = **11.443**

C_6" = 1 1 0 1 0 1 **1** 0 0 **0** 0 0 0 0 0 1 **1** 0 0 0 0 **1** 0 0 1 **0** 1 1 1 1 0 0 1
= **{ 219142** ; **2425 }** = **35.340**

C_7" = **1** 0 0 **1** 0 1 0 1 1 0 0 1 0 1 0 1 1 0 **1** 0 0 **1** 0 1 **1** 1 1 0 1 1 1 1 0
= { 153174 ; 19422 } = 19.229

C_8" = 1 1 0 0 1 0 1 1 0 0 0 0 1 0 0 0 1 1 1 0 1 **0** 1 0 0 1 1 **0** 1 1 1 1 0
= { 207907 ; 21726 } = 24.000

C_9" = 0 1 **0** 0 1 0 0 0 0 0 1 1 1 1 0 **0** 1 1 0 **1** 0 0 1 **0** **1** 1 1 1 0 0 1 0 **0**
= { 73971 ; 9700 } = 22.262

C_{10}" = 0 **1** 1 0 0 0 0 1 1 0 1 1 **1** 1 0 0 0 **0** 0 1 1 1 1 1 1 **1** 1 0 1 0 0 0 0
= { 100080 ; 16336 } = 21.751

Gen yang mengalami proses *mutation* pada generasi ini adalah sebanyak 35. Terlihat bahwa semua *chromosome* pada tingkat probabilitas 10% harus melewati proses *mutation*. *Chromosome* 6 sekarang menjadi yang terkuat dan *chromosome* 5 tetap yang terlemah.

A.4. Proses Evaluasi

Secara umum, proses evaluasi dalam pencarian nilai fungsi sinus maksimum terdiri dari tiga tahapan, yaitu:

- Konversi *chromosome* (kode *binary*) ke dalam solusi yang dinyatakan dengan bilangan desimal.

$$(x_1)_i = \sum_{j=1}^{m} (c_{ij} \cdot 2^{(m-j)})$$

$$(x_2)_i = \sum_{j=m+1}^{m+n} (c_{ij} \cdot 2^{(m+n-j)})$$

dimana: x_1, x_2 = solusi dalam bilangan desimal

i = nomor *chromosome*

m = jumlah *gen* yang merepresentasikan solusi *x1*

n = jumlah *gen* yang merepresentasikan solusi *x2*

cij = nilai *gen* ke *j* dari *chromosome i* yang masih dalam kode *binary*

- Perhitungan nilai fungsi sinus , y = f (x_1,x_2), dengan menggunakan persamaan (1) berdasarkan nilai *x1* dan *x2* yang diperoleh untuk masing-masing *chromosome*.
- Perhitungan nilai kecocokan (*fitness*) yang dalam hal ini juga adalah y = f ‘ (x_1,x_2) , Untuk setiap generasi, nilai *y’max* selalu dapat diperoleh. Akan tetapi, apakah nilai *y’max* ini merupakan nilai fungsi maksimum global yang dicari tetap tidak dapat langsung diketahui dengan algoritma genetika. Untuk itu, proses iterasi harus terus dilakukan sampai semua generasi yang diinginkan (~ 1000 generasi) telah selesai dianalisis.

A.5. Proses Seleksi

Proses seleksi dilakukan untuk memilih *chromosome* induk dan *chromosome* turunan berdasarkan nilai kecocokan yang diperoleh di atas untuk membentuk generasi baru yang lebih baik ke arah solusi optimum yang dicari. Ada tiga ketentuan dasar yang harus dipertimbangkan dalam melakukan proses seleksi, yaitu:

a. Jumlah *chromosome* pada setiap generasi baru harus selalu dipertahankan tetap. (untuk contoh proses pencarian nilai fungsi sinus maksimum, jumlah *chromosome* = 10)
b. Fungsi seleksi yang cocok harus dipilih sesuai dengan jenis aplikasinya. Banyak fungsi seleksi yang telah diusulkan; Gen, et.al. (2005) mengelompokkan fungsi seleksi ke dalam tiga kategori, yaitu:
    - Fungsi seleksi normal yang cenderung untuk mengeliminasi *chromosome* yang memiliki nilai kecocokan ekstrim.
    - Fungsi seleksi direksional yang ditargetkan untuk menurunkan atau menaikkan nilai kecocokan rata-rata dari seluruh *chromosome* yang dianalisis (populasi).

- Fungsi seleksi acak yang cenderung untuk mengeliminasi *chromosome* yangmemiliki nilai kecocokan tengah.

Fungsi seleksi direksional merupakan fungsi yang umum digunakan, termasuk untuk contoh proses pencarian nilai fungsi sinus maksimum. Empat alternatif fungsi seleksi direksional dapat dianalisis dengan menggunakan program GA, termasuk:

1. 10 *chromosome* yang membentuk generasi baru dipilih secara random berdasarkan kurva distribusi kumulatif nilai fungsi kecocokan, baik yang berasal dari *chromosome* induk, maupun dari *chromosome* turunan dengan prosedur, sbb.:

   i) hitung probabilitas nilai fungsi kecocokan,

   $$P_i\ (i = 1 \div 20) = (\sum_{j=1}^{i} y'_j) / (\sum_{j=1}^{20} y'_j)$$

   ii) ambil nilai random, *Prandom (= 0 ÷ 1)*

   iii) pilih *chromosome* i yang memenuhi kondisi (*Pi-1 < Prandom ≤ Pi*)

   iv) ulangi langkah ii) dan iii) untuk 10 *chromosome* yg membentuk generasi baru

2. Sama seperti alternatif 1 di atas tetapi *chromosome* terbaik dari generasi induk atau generasi turunan harus selalu disertakan dalam generasi baru.
3. Semua *chromosome* turunan dipilih sebagai generasi baru, kecuali *chromosome* terburuk yang digantikan oleh *chromosome* terbaik dari generasi induk; dan untuk menghindari duplikasi, *chromosome* pengganti harus lebih baik dibandingkan dengan yang terbaik dari *chromosome* turunan.
4. 10 *chromosome* yang memiliki nilai fungsi kecocokan terbesar dipilih dari antara gabungan *chromosome* induk dan *chromosome* turunan sebagai generasi baru.

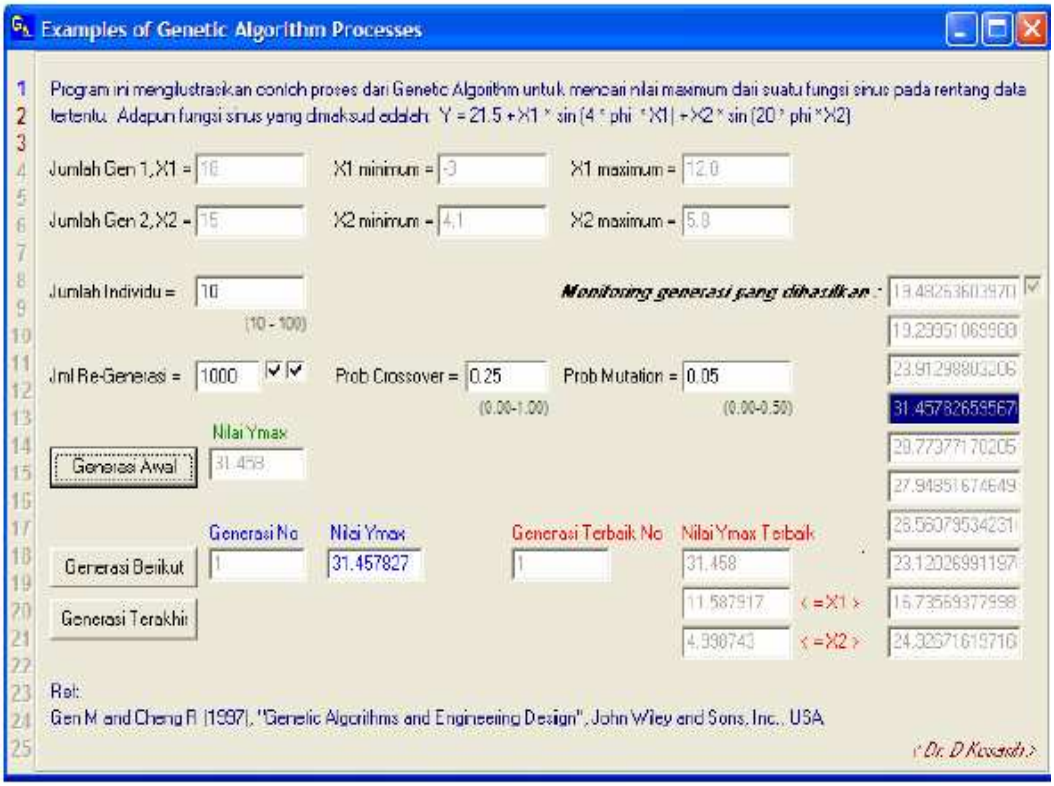
c. Duplikasi *chromosome* pada generasi baru harus dapat dicegah untuk menghindari proses pencarian terperangkap pada solusi optimum lokal. Disamping itu, nilai fungsi *chromosome* yang saling berdekatan juga tidak disukai karena akan mempersempit ruang explorasi.

Pada contoh, generasi baru yang dihasilkan dari proses seleksi untuk fungsi seleksi direksional alternatif 3 adalah sama dengan generasi turunan (pada Bab III.3), karena *chromosome* terbaik dari generasi induk tidak lebih baik dibandingkan dengan

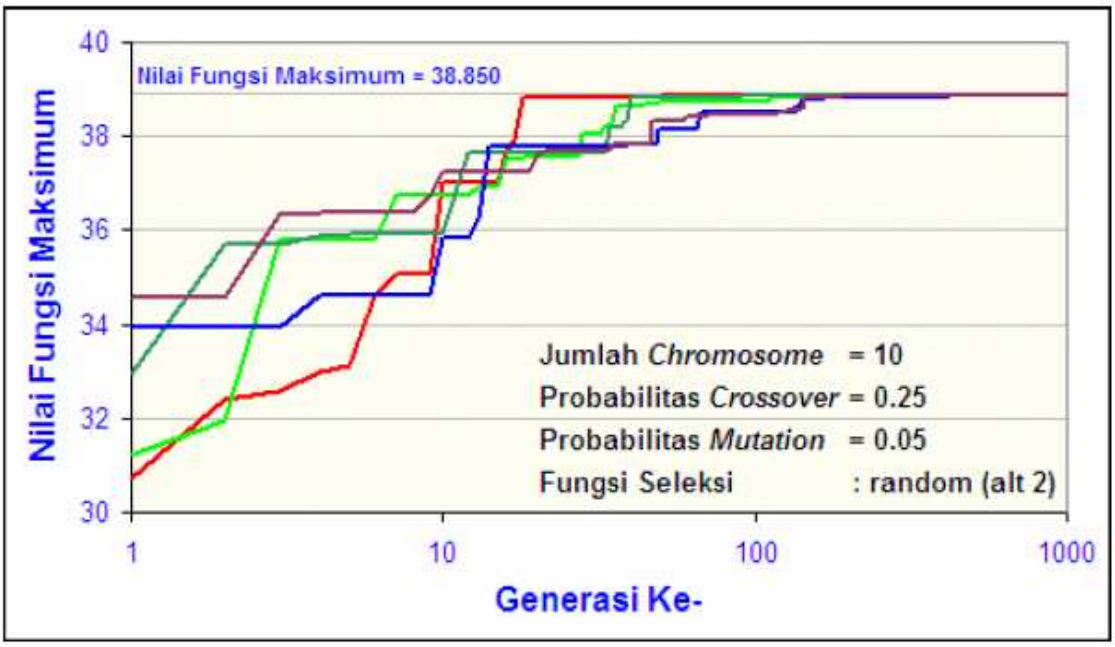
chromosome terbaik dari generasi turunan. Jadi, nilai fungsi sinus terbaik yang dihasilkan pada generasi kedua ini adalah tetap 35.340 pada *chromosome* 6. Sedangkan, nilai fungsi sinus maksimum yang diharapkan adalah 38.850, untuk nilai *x1* = 11.6255 dan *x2* = 5.7250. Untuk itu, proses regenerasi harus terus dilanjutkan, seperti yang akan diuraikan berikut ini.

B. Program GA

Tampilan program GA untuk pencarian nilai fungsi sinus maksimum diperlihatkan pada Gambar dibawah. Proses regenerasi selalu dilakukan untuk 1000 generasi. Generasi yang menghasilkan *chromosome* (solusi) terbaik dapat diketahui di akhir proses regenerasi. Ketiga data input utama lainnya, yaitu jumlah *chromosome*, probabilitas *crossover* dan probabilitas *mutation* juga diperlihatkan di sini.



Gambar 3 : Contoh data input (*default*) pada program GA dengan hasil generasi awal



Gambar 4: Tahapan konvergensi dalam mencapai nilai fungsi sinus maksimum untuk 5x pengulangan proses

Gambar 4 memperlihatkan trayektori pencapaian generasi terbaik yang memberikan nilai fungsi sinus maksimum untuk 5 kali pengulangan proses. Sifat *stochastic* algoritma genetika mulai dari proses pembentukan generasi awal sampai pada proses pencapaian generasi terbaik yang selalu berubah terlihat pada Gambar 4. Bahkan, solusi optimum tidak selamanya akan dapat dihasilkan jika empat data input utama, yaitu jumlah *chromosome*, probabilitas *crossover*, probabilitas *mutation* dan fungsi seleksi tidak ditentukan secara tepat, seperti yang akan dibahas pada subbab C berikut.

C. Analisis Faktor-Faktor Pengaruh Dominan Dalam Algoritma Genetika

Gambar 5 memperlihatkan pengaruh dari masing-masing data input utama terhadap nilai fungsi sinus maksimum yang dihasilkan. Untuk menggambarkan keempat faktor pengaruh tersebut digunakan nilai data input berikut sebagai acuan, yang dari analisis pendahuluan diketahui merupakan data input yang relatif terbaik.

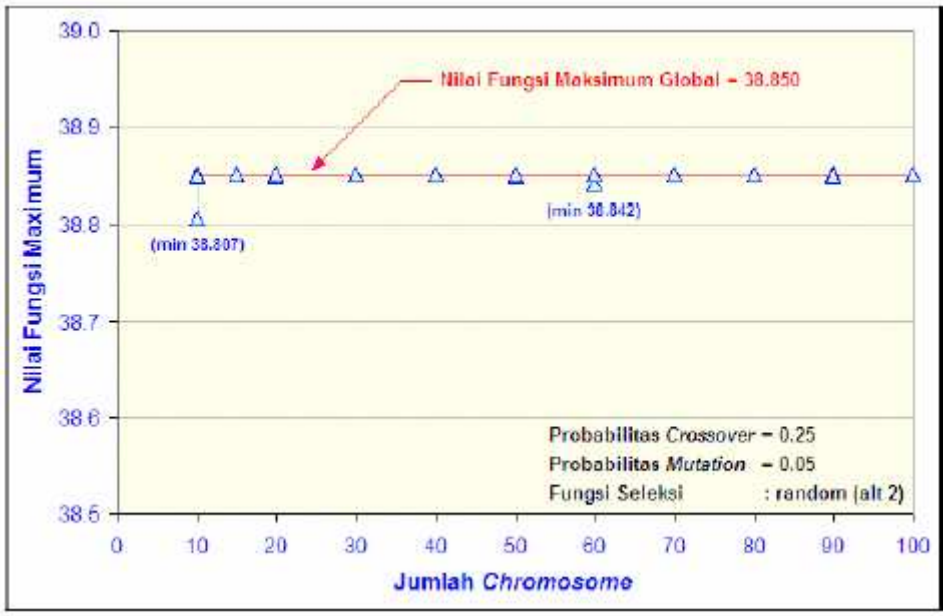
Jumlah *chromosome* = 10

Probabilitas *crossover* = 0.25

Probabilitas *mutation* = 0.05

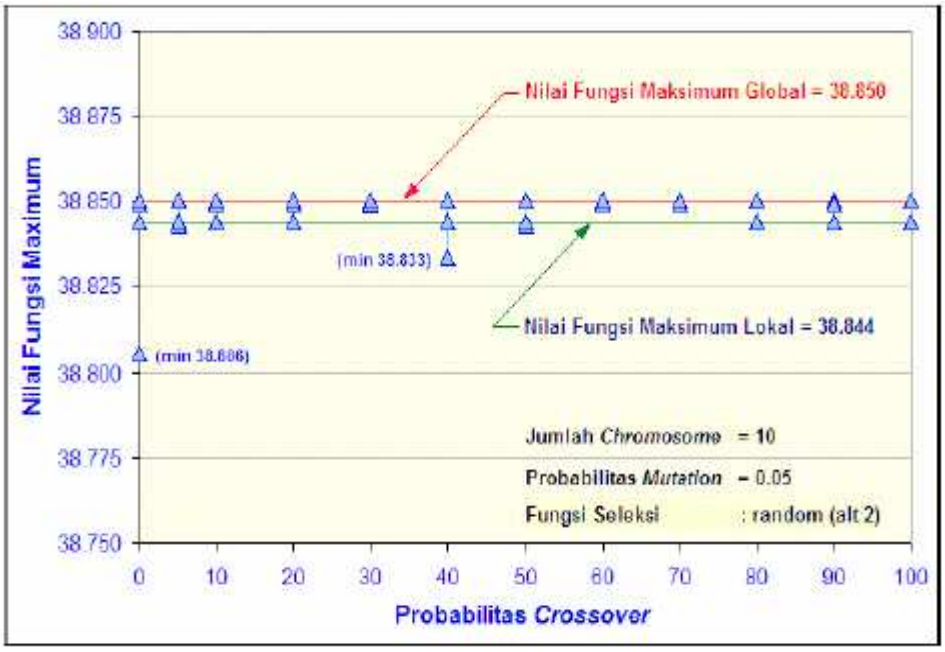
Fungsi seleksi : memilih *chromosome* turunan secara random sebagai generasi baru dan harus selalu menyertakan *chromosome* yang memberikan nilai fungsi sinus maksimum (alternatif 2)

Gambar 5a memperlihatkan bahwa rentang jumlah *chromosome* yang dianalisis bervariasi dari 10 ÷ 100. Secara umum, nilai fungsi sinus maksimum selalu dapat diperoleh dalam 5 kali pengulangan proses, kecuali pada jumlah *chromosome* sebesar 10 dan 60 dimana solusi yang diperoleh agak bervariasi. Sedikit perbedaan solusi juga terlihat pada jumlah *chromosome* sebesar 20, 50 dan 90. Perbedaan2 solusi tersebut pada prinsipnya selalu mungkin terjadi dalam proses pencarian nilai optimum dengan menggunakan algoritma genetika yang berdasarkan pendekatan *stochastic*. Meskipun demikian, jika siklus regenerasi diperpanjang, maka nilai fungsi sinus maksimum umumnya akan dapat diperoleh.

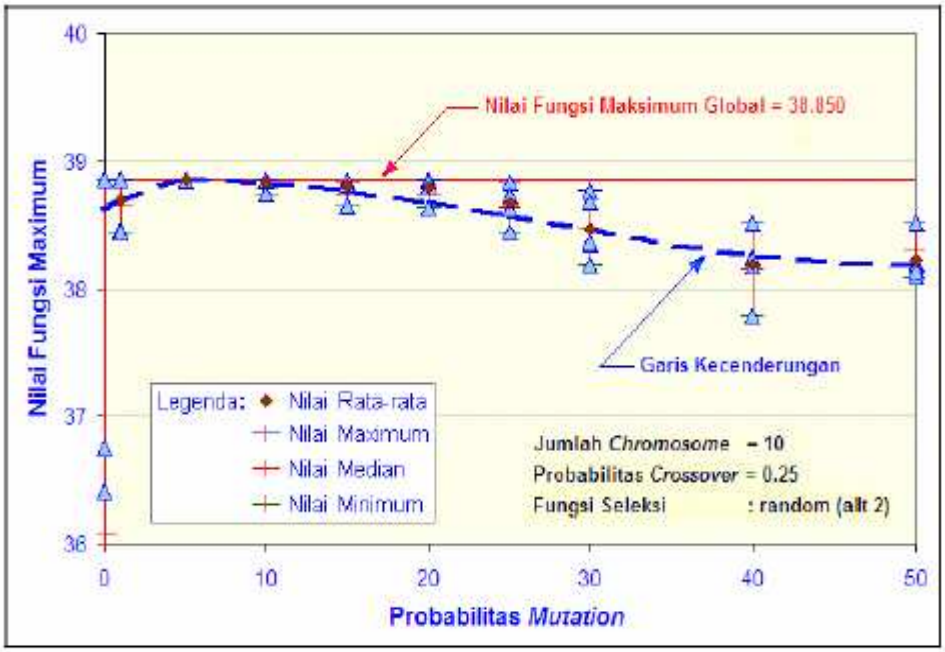


(a) pengaruh dari jumlah *chromosome*

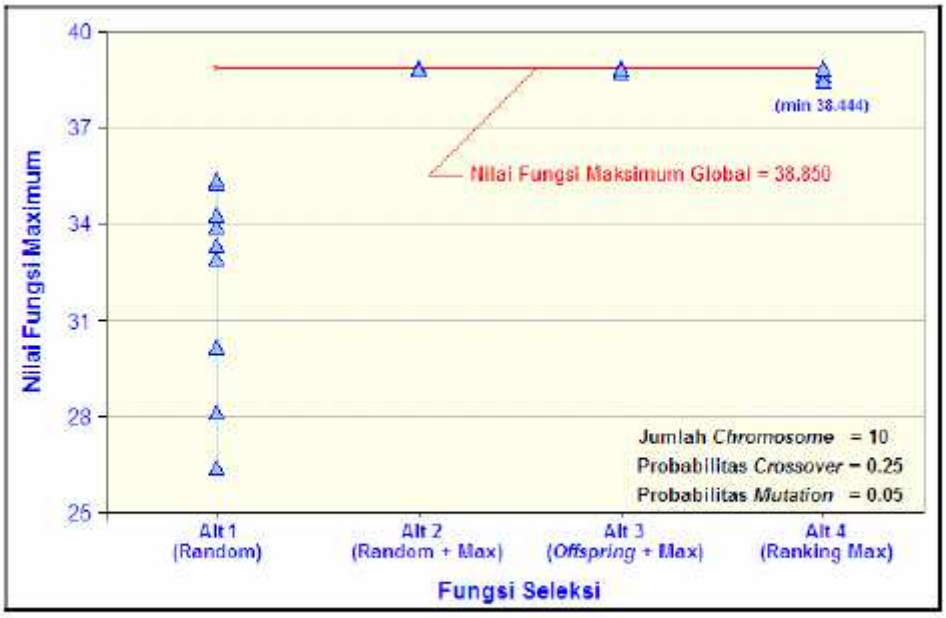
Gambar 5: Pengaruh dari berbagai data input terhadap nilai fungsi sinus maksimum dalam 5x pengulangan proses



(b) pengaruh dari probabilitas *crossover*



(c) pengaruh dari probabilitas *mutation*



(d) pengaruh dari fungsi seleksi

Gambar 5: Pengaruh dari berbagai data input terhadap nilai fungsi sinus maksimum dalam 5x pengulangan proses (...lanjutan)

Rentang probabilitas *crossover* yang dianalisis bervariasi dari 0 ÷ 100%. Gambar 5b menunjukkan, bahwa secara umum nilai fungsi sinus maksimum juga tidak begitu dipengaruhi oleh tingkat probabilitas *crossover*. Variasi nilai fungsi sinus maksimum yang terlihat pada Gambar 5b akan berkurang jika jumlah *chromosome* yang dianalisis diperbesar.

Yang cukup menarik dari analisis sensitivitas terhadap tingkat probabilitas *crossover* adalah bahwa proses pencarian nilai optimum seringkali terperangkap pada nilai fungsi sinus maksimum lokal sebesar 38.844. Dari analisis awal diketahui, bahwa masalah ini kelihatannya dapat diatasi dengan menggunakan fungsi seleksi gabungan. Studi yang lebih mendalam masih diperlukan untuk menjelaskan kasus ini.

Pengaruh tingkat probabilitas *mutation* terhadap nilai fungsi sinus maksimum ternyata sangat signifikan, seperti diperlihatkan pada Gambar 5c. Bahkan, jika proses *mutation* tidak dilakukan, maka proses pencarian nilai optimum dengan menggunakan algoritma genetika sulit untuk konvergen. Untuk contoh kasus yang sedang dianalisis, tingkat probabilitas *mutation* terbaik kurang lebih adalah 5%.

Akhirnya, Gambar 5d memperlihatkan pengaruh fungsi seleksi terhadap nilai fungsi sinus maksimum. Generasi baru yang ditentukan hanya secara random (alternatif 1) umumnya tidak dapat memberikan solusi optimum. Sebaliknya, solusi optimum hampir

selalu dapat diperoleh dengan menggunakan pendekatan *meta-heuristic* yang terus mempertahankan *chromosome* terbaik pada setiap proses regenerasi (alternatif 2 ÷ 4); dan, alternatif 2 yang merupakan pendekatan *stochastic* secara marginal dapat dikatakan lebih menjanjikan untuk menghasilkan solusi optimum yang diinginkan dalam kasus ini.

REFERENSI

Dari berbagai sumber