Modul Kuliah
Semester Ganjil 2015-2016

DASAR-DASAR ALGORITMA EVOLUSI

Wayan Firdaus Mahmudy

Program Teknologi Informasi dan Ilmu Komputer (PTIIK)

Universitas Brawijaya

Kata Pengantar

Algoritma evolusi berkembang seiring dengan kemajuan perangkat lunak pemrograman. Kemampuannya untuk menyelesaikan berbagai permasalahan kompleks menjadikan algoritma ini banyak dipilih untuk berbagai kasus optimasi. Buku modul kuliah ini memberikan dasar-dasar yang diperlukan oleh mahasiswa untuk menerapkan algoritma evolusi dalam berbagai permasalahan di dunia nyata.

Buku ini disusun sebagai bahan penunjang untuk pengajaran mata kuliah Algoritma Evolusi di Program Teknologi Informasi dan Ilmu Komputer (PTIIK), Universitas Brawijaya. Buku ini juga dimaksudkan untuk mengisi kelangkaan materi dalam Bahasa Indonesia yang secara mendetail membahas penerapan algoritma evolusi dalam berbagai bidang riset. Berbagai referensi dari jurnal ilmiah nasional dan internasional diacu untuk memberikan gambaran tren terkini dalam pengembangan dan penerapan algoritma evolusi.

Studi kasus dari berbagai riset di literatur diberikan untuk membantu pemahaman pembaca bagaimana menerapkan algoritma evolusi untuk masalah yang sederhana sampai masalah yang kompleks. Selain dimaksudkan sebagai buku pegangan kuliah bagi mahasiswa S1, buku ini bisa dijadikan acuan bagi mahasiswa yang mengerjakan skripsi atau tugas akhir.

Beberapa artikel penulis (jurnal dan konferensi) yang diacu dalam buku ini bisa didapatkan pada https://wayanfm.lecture.ub.ac.id/research-publications/.

Malang, Agustus 2015

Daftar Isi

KATA PENGANTAR ..... **I**

DAFTAR ISI ..... **II**

DAFTAR GAMBAR ..... **V**

DAFTAR SINGKATAN ..... **VII**

BAB 1 ..... **1**

TEKNIK OPTIMASI 1

1.1. Pengantar 1

1.2. Klasifikasi Teknik Optimasi 2

1.3. Prinsip Kerja Algoritma Evolusi 4

1.4. Rangkuman 5

1.5. Latihan 6

BAB 2 ..... **7**

DASAR-DASAR ALGORITMA GENETIKA 7

2.1. Pengantar 7

2.2. Struktur Algoritma Genetika 9

2.3. Studi Kasus: Maksimasi Fungsi Sederhana 11

2.3.1. Inisialisasi 11

2.3.2. Reproduksi 12

2.3.3. Evaluasi 13

2.3.4. Seleksi 14

2.4. Studi Kasus: Maksimasi Fungsi dengan Presisi Tertentu 15

2.4.1. Representasi Chromosome 16

2.4.2. Inisialisasi 17

2.4.3. Reproduksi 18

2.4.4. Evaluasi 19

2.4.5. Seleksi 20

2.5. Kondisi Berhenti (*Termination Condition*) 25

2.6. Rangkuman 25

2.7. Latihan 26

BAB 3 ..... **29**

ALGORITMA GENETIKA DENGAN PENGKODEAN REAL (*REAL-CODED GA/RCGA*) 29

3.1. Siklus RCGA 29

3.1.1. Representasi Chromosome 29

3.1.2. Inisialisasi 29

3.1.3. Reproduksi 30

3.1.4. Seleksi 31

3.2. Alternatif Operator Reproduksi pada Pengkodean Real 34
3.3. Alternatif Operator Seleksi 35
3.3.1. Elitism Selection 35
3.3.2. Replacement Selection 36
3.4. Diskusi: Nilai Parameter Algoritma Genetika 38
3.5. Diskusi: Mekanisme Sampling Proses Seleksi 38
3.6. Diskusi: Probabilitas Seleksi 39
3.7. Diskusi: Penanganan Konvergensi Dini 41
3.8. Rangkuman 44
3.9. Latihan 44
BAB 4 47
OPTIMASI MASALAH KOMBINATORIAL 47
4.1. Pengantar 47
4.2. Travelling Salesman Problem (TSP) 47
4.2.1. Representasi Chromosome 48
4.2.2. Crossover 49
4.2.3. Mutasi 50
4.3. Flow-Shop Scheduling Problem (FSP) 50
4.4. Two-Stage Assembly Flow-Shop Scheduling Problem 51
4.5. Job-Shop Scheduling Problem (JSP) 53
4.5.1. Representasi Chromosome 53
4.5.2. Crossover 53
4.5.3. Mutasi 54
4.5.4. Representasi Permutasi Untuk JSP 54
4.6. Transportation Problem 54
4.6.1. Representasi Chromosome 56
4.6.2. Crossover 57
4.6.3. Mutasi 58
4.6.4. Representasi Permutasi 58
4.7. Flexible Job-Shop Scheduling Problem (FJSP) 60
4.7.1. Representasi Chromosome *Task Sequencing List* 61
4.7.2. Representasi Chromosome Bilangan Pecahan 62
4.8. Multi Travelling Salesman Problem (m-TSP) 64
4.9. Vehicle Routing Problem With Time Window (VRPTW) 65
4.10. Rangkuman 66
4.11. Latihan 67
BAB 5 69
TOPIK LANJUT PADA ALGORITMA GENETIKA 69
5.1. Pengantar 69
5.2. Hybrid Genetic Algorithms (HGAs) 69
5.3. Parallel Genetic Algorithms (PGAs) 71

5.4. Nilai Parameter Adaptif 72
5.5. Rangkuman 74
5.6. Latihan 74

BAB 6 75

EVOLUTION STRATEGIES (ES) 75
6.1. Pengantar 75
6.2. Struktur Dasar Evolution Strategies 76
6.3. Siklus ES (μ, λ) 77
6.3.1. Representasi Chromosome 77
6.3.2. Inisialisasi 77
6.3.3. Reproduksi 78
6.3.4. Seleksi 79
6.4. Siklus ES (μ/r + λ) 79
6.4.1. Reproduksi: Recombinasi dan Mutasi 79
6.4.2. Seleksi 80
6.5. Studi Kasus ES (μ + λ): Optimasi Fungsi Berkendala 81
6.5.1. Inisialisasi 82
6.5.2. Reproduksi 83
6.5.3. Seleksi 83
6.6. ES untuk Representasi Permutasi 84
6.7. Rangkuman 85
6.8. Latihan 85

BAB 7 87

GENETIC PROGRAMMING (GP) DAN EVOLUTIONARY PROGRAMMING (EP) 87
7.1. Genetic Programming 87
7.2. Siklus Genetic Programming 89
7.2.1. Representasi Chromosome 89
7.2.2. Inisialisasi dan Evaluasi 89
7.2.3. Crossover 90
7.2.4. Mutasi 91
7.2.5. Seleksi 91
7.3. Evolutionary Programming (EP) 92
7.4. Studi Kasus 1: Pohon Keputusan 92
7.4.1. Representasi Chromosome 93
7.4.2. Inisialisasi dan Evaluasi 94
7.4.3. Crossover, Mutasi, dan Seleksi 94
7.5. Rangkuman 95
7.6. Latihan 95

DAFTAR PUSTAKA 97

INDEKS 104

Daftar Gambar

GAMBAR 1.1. POSISI EA DI ANTARA TEKNIK OPTIMASI LAIN 3

GAMBAR 1.2. PROSES-PROSES DALAM EAS 4

GAMBAR 2.1. MENCARI SOLUSI DENGAN ALGORITMA GENETIKA 10

GAMBAR 2.2. GRAFIK FUNGSI CONTOH (2.1) 11

GAMBAR 2.3. SIKLUS ALGORITMA GENETIKA 15

GAMBAR 2.4. PLOTTING 2D FUNGSI CONTOH (2.2) 16

GAMBAR 2.5. ROULETTE WHEEL PELUANG TERPILIHNYA SETIAP INDIVIDU 22

GAMBAR 2.6. SOLUSI GA TIAP GENERASI UNTUK FUNGSI UJI (2.2) 24

GAMBAR 3.1. PSEUDO-CODE BINARY TOURNAMENT SELECTION 32

GAMBAR 3.2. SOLUSI RCGA PADA TIAP GENERASI 33

GAMBAR 3.3. PSEUDO-CODE *ELITISM SELECTION* 35

GAMBAR 3.4. PSEUDO-CODE *REPLACEMENT SELECTION* 37

GAMBAR 3.5. SOLUSI RCGA PADA TIAP GENERASI MENGGUNAKAN RANDOM INJECTION SATU INDIVIDU 42

GAMBAR 3.6. PSEUDO-CODE PERHITUNGAN RASIO KEMIRIPAN DUA KROMOSOM 43

GAMBAR 3.7. PSEUDO-CODE PERHITUNGAN NILAI KERAGAMAN POPULASI 43

GAMBAR 4.1. CONTOH MASALAH TSP 48

GAMBAR 4.2. CROSSOVER PADA REPRESENTASI PERMUTASI 49

GAMBAR 4.3. RECIPROCAL EXCHANGE MUTATION 50

GAMBAR 4.4. INSERTION MUTATION 50

GAMBAR 4.5. GANTT-CHART UNTUK URUTAN JOB J1 → J2 → J3 51

GAMBAR 4.6. GANTT-CHART UNTUK URUTAN JOB J2 → J1 → J3 51

GAMBAR 4.7. GANTT-CHART UNTUK TWO-STAGE ASSEMBLY FLOWSHOP 52

GAMBAR 4.8. GANTT-CHART UNTUK JSP 53

GAMBAR 4.9. CROSSOVER PADA *JOB-BASED REPRESENTATION* 54

GAMBAR 4.10. REPRESENTASI PERMUTASI UNTUK JSP 54

GAMBAR 4.11. MODEL TRANSPORTASI 55

GAMBAR 4.12. CONTOH CHROMOSOME DARI *TASK SEQUENCING LIST REPRESENTATION* .. 62

GAMBAR 4.13. GANT-CHART UNTUK MENGHITUNG MAKESPAN 64

GAMBAR 4.14. CONTOH CHROMOSOME UNTUK MTSP 64

GAMBAR 4.15. CONTOH SOLUSI MTSP UNTUK 40 NODE 65

GAMBAR 5.1. MAS DAN OPTIMASI LOKAL (GEN & CHENG 2000) 70
GAMBAR 5.2. MEKANISME MIGRASI 71
GAMBAR 5.3. MEKANISME PENGUBAHAN MUTATION RATE SECARA ADAPTIF 73
GAMBAR 7.1. BINARY TREE SEBUAH FUNGSI NON-LINEAR 88
GAMBAR 7.2. CONTOH DUA INDIVIDU RANDOM 89
GAMBAR 7.3. CONTOH PROSES CROSSOVER 91
GAMBAR 7.4. CONTOH PROSES MUTASI 91
GAMBAR 7.5. POHON KEPUTUSAN PENERIMAAN PENGAJUAN KREDIT 93
GAMBAR 7.6. CONTOH DUA INDIVIDU RANDOM 94
GAMBAR 7.7. CONTOH MUTASI MENGUBAH ANGKA 95

Daftar Singkatan

ACO	Ant Colony Optimization
EAs	Evolutionary Algorithms
EC	Evolutionary Computation
EP	Evolutionary Programming
ES	Evolution Strategies
FJSP	Flexible Job-Shop Problem
FSP	Flow-Shop Problem
GAs	Genetic Algorithms
GP	Genetic Programming
HGAs	Hybrid Genetic Algorithms
JSP	Job-Shop Problem
LS	Local Search
MAs	Memetic Algorithms
PSO	Particle Swarm Optimization
RCGA	Real Coded Genetic Algorithm
SA	Simulated Annealing
SPT	Shortest Processing Time
TS	Tabu Search
TSP	Travelling Salesman Problem
VNS	Variable Neighbourhoods Search

BAB 1

Teknik Optimasi

1.1. Pengantar

Dalam kehidupan sehari-hari seringkali kita berhadapan dengan pencarian solusi suatu masalah seperti contoh berikut:

- Pembuatan jadwal kuliah yang mencakup ketersediaan dosen dan ruangan. Jadwal harus dibuat tujuan untuk menghindari seorang dosen/mahasiswa terjadwal di lebih dari satu kelas pada waktu yang sama (Liliana & Mahmudy 2006). Penjadwalan juga bisa diterapkan pada penyusunan jadwal ujian (Mawaddah & Mahmudy 2006), jadwal jaga perawat (Ilmi, Mahmudy & Ratnawati 2015) atau mata pelajaran di sekolah (Sari, DDP, Mahmudy & Ratnawati 2015).
- Persoalan transportasi yang mencakup pendistribusian suatu komoditas atau produk dari sejumlah sumber kepada sejumlah tujuan dengan tujuan meminimumkan ongkos pengangkutan yang terjadi (Munawaroh & Mahmudy 2015a; Panharesi & Mahmudy 2015; Putri, FB, Mahmudy & Ratnawati 2015).
- Pemilihan rute terpendek (biaya terkecil) untuk mengunjungi sejumlah kota (Endarwati, Mahmudy & Ratnawati 2014; Pitaloka, Mahmudy & Sutrisno 2014).
- Penentuan komposisi makanan baik bagi manusia maupun ternak dengan biaya minimum yang harus memenuhi batasan minimal untuk setiap komponen nutrisi (Pratiwi, Mahmudy & Dewi 2014; Rianawati & Mahmudy 2015; Sari, AP, Mahmudy & Dewi 2014; Wahid & Mahmudy 2015).

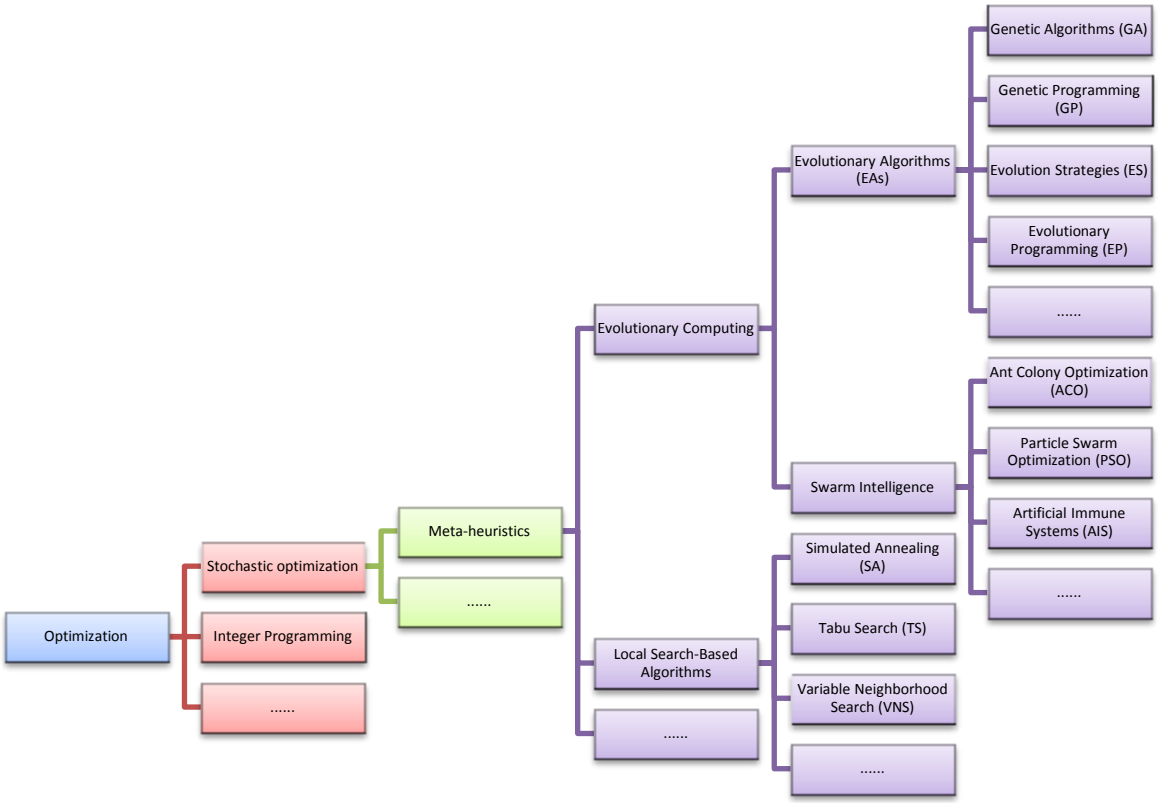
Penyelesaian masalah di atas akan mudah dilakukan jika ukuran data relatif kecil. Masalah akan menjadi kompleks jika data berukuran besar atau melibatkan sejumlah besar entitas. Pada masalah kompleks dibutuhkan juga formulasi matematika yang kompleks yang bisa jadi sangat sulit dibangun atau membutuhkan waktu yang lama. Berdasarkan model matematis yang dibangun bisa dilakukan analisis untuk mencari solusi yang terbaik

(optimum). Solusi optimum mungkin dapat diperoleh tetapi memerlukan proses perhitungan yang panjang.

Untuk menyelesaikan kasus khusus seperti di atas dapat digunakan metode heuristik, yaitu suatu metode pencarian yang didasarkan atas intuisi atau aturan-aturan empiris untuk memperoleh solusi yang lebih baik daripada solusi yang telah dicapai sebelumnya. Metode ini tidak selalu menghasilkan solusi optimum tetapi jika dirancang dengan baik akan menghasilkan solusi yang mendekati optimum dalam waktu yang relatif cepat. Metode heuristis yang bisa diterapkan pada masalah optimasi misalnya algoritma koloni semut (Irwansyah, Pinandito & Mahmudy 2014), particle swarm optimization (Mahmudy 2014c, 2015b), algoritma hill-climbing (Achnas, Cholissodin & Mahmudy 2015; Mahmudy 2008b), algoritma A* (Nurhumam & Mahmudy 2008), tabu search (Seputro, Mahmudy & Mursityo 2015), algoritma simulated annealing (Mahmudy 2014b), variable neighbourhood search (Mahmudy 2015a), dan algoritma evolusi .

1.2. Klasifikasi Teknik Optimasi

Algoritma evolusi (*evolutionary algorithms, EAs*) merupakan sub-set dari komputasi evolusi (*evolutionary computation, EC*) yang merupakan bentuk generik dari algoritma optimasi *meta-heuristic* berbasis populasi. Secara umum, posisi dari EAs di antara teknik optimasi lainnya ditunjukkan pada Gambar 1.1. Tergantung dari kriteria yang digunakan untuk pengklasifikasian, struktur ini bisa berubah.



Gambar 1.1. Posisi EA di antara teknik optimasi lain

Pada Gambar 1.1, *optimization* didefinisikan sebagai proses pemilihan sebuah solusi dari sejumlah alternative solusi dengan memenuhi sejumlah batasan (*contstraints*). Misalkan pada pencarian rute untuk mengunjungi sejumlah kota. Pada kasus ini tentu saja terdapat banyak alternative pilihan rute (solusi). Solusi yang dipilih disesuaikan dengan tujuan (*objective*) dari permasalahan ini, misalkan memilih rute terpendek atau rute dengan waktu tempuh tercepat. Batasan yang ada misalkan setiap kota harus dikunjungi tepat satu kali.

Stochastic optimization menggunakan bilangan acak (*random*) dalam pencarian solusi. Sebagai konsekuensinya, sebuah algoritma dalam kelas ini setiap dijalankan akan menghasilkan solusi akhir yang berbeda, meskipun diterapkan dalam permasalahan yang sama.

Sebuah algoritma *meta-heuristic* bertindak sebagai 'manajer' dari beberapa algoritma *heuristic* untuk secara terorganisir mencari solusi dari sebuah permasalahan. Misalkan

metode *Variable Neighborhood Search* (VNS) yang me-*manage* sebuah teknik *local search* (LS). VNS secara sistematis meng-*iterasi* LS untuk mencari solusi dari titik awal yang berbeda serta mencakup area pencarian yang lebih luas . Contoh lainnya adalah algoritma genetika yang me-*manage* beberapa *genetic operator* seperti *crossover*, *mutation*, dan s*election*. Materi pada Bab 2 akan menjelaskan hal ini secara mendetail.

Evolutionary computing merujuk kepada berbagai teknik penyelesaian masalah yang berbasis proses evolusi biologi seperti seleksi alam (*natural selection*) dan penurunan sifat genetis (*genetic inheritance*). Berbagai teknik dalam kelas ini telah diaplikasikan pada berbagai permasalahan praktis. Salah satu sub-kelas dari *evolutionary computing* adalah algoritma evolusi yang sedang anda pelajari.

1.3. Prinsip Kerja Algoritma Evolusi

Algoritma Evolusi (*evolutionary algorithms*, EAs) merupakan teknik optimasi yang meniru proses evolusi biologi. Menurut teori evolusi terdapat sejumlah individu dalam populasi. Dari generasi ke generasi, individu-individu ini berperan sebagai induk (*parent*) yang melakukan reproduksi menghasilkan keturunan (*offspring*). Individu-individu ini (beserta *offspring*) berevolusi dan individu-individu yang lebih baik (mampu beradaptasi dengan lingkungannya) mempunyai peluang lebih besar untuk melewati seleksi alam (*natural selection*) dan bertahan hidup. Individu yang lebih baik juga cenderung (tidak selalu tapi mempunyai kemungkinan lebih besar) menghasilkan keturunan yang lebih baik sehingga dari generasi ke generasi akan terbentuk populasi yang lebih baik. Keseluruhan proses dalam EAs ditunjukkan pada Gambar 1.2.

Gambar 1.2. Proses-proses dalam EAs

Individu-individu dalam populasi di EAs merepresentasikan solusi dari masalah yang akan diselesaikan. Sebuah fungsi *fitness* digunakan untuk mengukur seberapa baik suatu

individu. Individu terbaik di akhir generasi bisa didekodekan sebagai solusi terbaik yang bisa diperoleh.

Dari penjelasan di atas, EAs bisa dikelompokkan dalam algoritma '*generate and test*' yang berbasis populasi (*population based*). EA juga bersifat *stochastic*, setiap kali dijalankan untuk masalah yang sama ada kemungkinan menghasilkan solusi yang berbeda (Smith & Eiben 2003).

Berbagai tipe EAs telah dikembangkan sebagai berikut:

- Algoritma genetika (*Genetic Algorithms*, GAs), merupakan tipe EAs yang paling popular dan banyak diterapkan pada masalah-masalah kompleks. Pada awalnya banyak menggunakan representasi string biner tapi kemudian berkembang dengan menggunakan vektor bilangan integer dan pecahan (*real*). Pembangkitkan solusi baru banyak mengandalkan proses tukar silang (*crossover*). Mutasi biasanya dipakai sebagai operator tambahan untuk menjaga keragaman populasi.
- *Evolution Strategies* (ES), representasi solusi biasanya menggunakan vektor bilangan pecahan. Mutasi merupakan operator reproduksi utama. Mekanisme *self-adaptation* digunakan untuk mengontrol perubahan nilai parameter pencarian.
- *Genetic Programming* (GP), digunakan untuk mengoptimasi rangkaian program komputer yang direpresentasikan dalam bentuk struktur data pohon (*tree*).
- *Evolutionary Programming* (EP), mempunyai tujuan seperti GP tapi prinsip kerjanya seperti ES. Finite State Machines (FSM) digunakan untuk merepresentasikan program komputer.

1.4. Rangkuman

Pada bab ini telah dibahas tentang klasifikasi teknik optimasi dan pentingnya algoritma evolusi untuk penyelesaian masalah kompleks yang sulit dipecahkan secara analitis menggunakan model matematis.

1.5. Latihan

Untuk memperjelas pemahaman anda, kerjakanlah latihan berikut sebisa mungkin tanpa melihat materi pada buku!

1. Pada jenis permasalahan apa algoritma heuristik seharusnya diterapkan?
2. Jelaskan apa yang dimaksud dengan individu dalam algoritma evolusi!
3. Apa yang dimaksud dengan fungsi *fitness*?
4. Apa yang dimaksud dengan pernyataan bahwa algoritma evolusi bersifat *stochastic*?

BAB 2

Dasar-Dasar Algoritma Genetika

2.1. Pengantar

Algoritma genetika (*Genetic Algorithms, GAs*) merupakan tipe EA yang paling popular. Algoritma genetika berkembang seiring dengan perkembangan teknologi informasi yang sangat pesat. Karena kemampuannya untuk menyelesaikan berbagai masalah kompleks, algoritma ini banyak digunakan dalam bidang fisika, biologi, ekonomi, sosiologi dan lain-lain yang sering menghadapi masalah optimasi yang model matematikanya kompleks atau bahkan sulit dibangun.

Dalam bidang industri manufaktur, GAs digunakan untuk perencanaan dan penjadwalan produksi (Mahmudy, Marian & Luong 2012a, 2013a). GA juga bisa diterapkan untuk kompresi citra (Ciptayani, Mahmudy & Widodo 2009), optimasi penugasan mengajar bagi dosen (Mahmudy 2006), penjadwalan dan alokasi ruang ujian (Mawaddah & Mahmudy 2006), optimasi penjadwalan kuliah (Liliana & Mahmudy 2006), optimasi *multi travelling salesman problem* (M-TSP) (Mahmudy 2008a), distribusi produk (Sulistiyorini & Mahmudy 2015), prediksi harga saham (Rahmi, Mahmudy & Setiawan 2015), optimasi lahan pertanian (Saputro, Mahmudy & Dewi 2015), optimasi laba produksi (Samaher & Mahmudy 2015), penentuan komposisi pakan ternak (Kusuma, Mahmudy & Indriati 2015), optimasi persediaan barang (Ramuna & Mahmudy 2015), pembentukan model regresi (Permatasari & Mahmudy 2015), pembentukan fungsi keanggotaan fuzzy (Azizah, Cholissodin & Mahmudy 2015), dan penyusunan rute dan jadwal kunjungan wisata yang efisien (Widodo & Mahmudy 2010).

Algoritma genetika diilhami oleh ilmu genetika, karena itu istilah yang digunakan dalam algoritma genetika banyak diadopsi dari ilmu tersebut. Apabila dibandingkan dengan prosedur pencarian dan optimasi biasa, algoritma genetika berbeda dalam beberapa hal sebagai berikut (Michalewicz 1996):

- Manipulasi dilakukan terhadap kode dari himpunan parameter (biasa disebut *chromosome*), tidak secara langsung terhadap parameternya sendiri.
- Proses pencarian dilakukan dari beberapa titik dalam satu populasi, tidak dari satu titik saja.
- Proses pencarian menggunakan informasi dari fungsi tujuan.
- Pencariannya menggunakan *stochastic operators* yang bersifat probabilistik, tidak menggunakan aturan deterministik.

Kelebihan GAs sebagai metode optimasi adalah sebagai berikut:

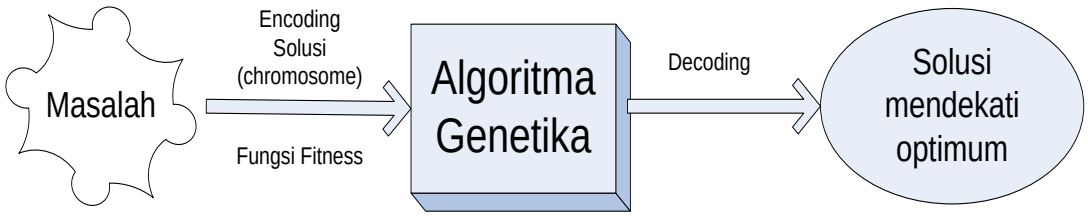
- GAs merupakan algoritma yang berbasis populasi yang memungkinkan digunakan pada optimasi masalah dengan ruang pencarian (*search space*) yang sangat luas dan kompleks. Properti ini juga memungkinkan GAs untuk melompat keluar dari daerah optimum lokal (Gen & Cheng 1997).
- Individu yang ada pada populasi bisa diletakkan pada beberapa sub-populasi yang diproses pada sejumlah komputer secara paralel. Hal ini bisa mengurangi waktu komputasi pada masalah yang sangat kompleks (Defersha & Chen 2010; Qi, Burns & Harrison 2000). Penggunaan sub-populasi juga bisa dilakukan pada hanya satu komputer untuk menjaga keragaman populasi dan meningkatkan kualitas hasil pencarian (Mahmudy 2009).
- GAs menghasilkan himpunan solusi optimal yang sangat berguna pada peyelesaian masalah dengan banyak obyektif (Mahmudy & Rahman 2011).
- GAs dapat digunakan untuk menyelesaikan masalah yang kompleks dengan banyak variabel. Variabel tersebut bisa kontinyu, diskrit atau campuran keduanya (Haupt & Haupt 2004).
- GAs menggunakan *chromosome* untuk mengkodekan solusi sehingga bisa melakukan pencarian tanpa memperhatikan informasi derivatif yang spesifik dari masalah yang diselesaikan (Gen & Cheng 1997; Haupt & Haupt 2004).

- GAs bisa diimplementasikan pada berbagai macam data seperti data yang dibangkitkan secara numerik atau menggunakan fungsi analitis (Haupt & Haupt 2004).
- GAs cukup fleksibel untuk dihibridisasikan dengan algoritma lainnya (Gen & Cheng 1997). Beberapa penelitian membuktikan bahwa *hybrid GAs* (*HGAs*) sangat efektif untuk menghasilkan solusi yang lebih baik (Mahmudy, Marian & Luong 2013e, 2013f, 2014).
- GAs bersifat *ergodic*, sembarang solusi bisa diperoleh dari solusi yang lain dengan hanya beberapa langkah. Hal ini memungkinkan eksplorasi pada daerah pencarian yang sangat luas dilakukan dengan lebih cepat dan mudah (Marian 2003).

2.2. Struktur Algoritma Genetika

Bagaimana menggunakan algoritma genetika untuk memecahkan suatu masalah ditunjukkan pada Gambar 2.1. Solusi dari suatu masalah harus dipetakan (*encoding*) menjadi string *chromosome*. String *chromosome* ini tersusun atas sejumlah *gen* yang menggambarkan variabel-variabel keputusan yang digunakan dalam solusi. Representasi string *chromosome* beserta fungsi *fitness* untuk menilai seberapa bagus sebuah *chromosome* (untuk menjadi solusi yang layak) dimasukkan ke algoritma genetika. Dalam banyak kasus, bagaimana merepresentasikan sebuah solusi menjadi *chromosome* sangat menentukan kualitas dari solusi yang dihasilkan (Mahmudy, Marian & Luong 2012a).

Dengan menirukan proses genetika dan seleksi alami maka algoritma genetika akan menghasilkan *chromosome* 'terbaik' setelah melewati sekian generasi. *Chromosome* 'terbaik' ini harus diuraikan (*decoding*) menjadi sebuah solusi yang diharapkan mendekati optimum.



Gambar 2.1. Mencari solusi dengan algoritma genetika

Apabila *P(t)* dan *C(t)* merupakan populasi (*parents)* dan *offspring* pada generasi ke-*t*, maka struktur umum algoritma genetika dapat dideskripsikan sebagai berikut (Gen & Cheng 1997):

```
procedure AlgoritmaGenetika
begin
  t = 0
  inisialisasi P(t)
  while (bukan kondisi berhenti) do
    reproduksi C(t) dari P(t)
    evaluasi P(t) dan C(t)
    seleksi P(t+1) dari P(t) dan C(t)
    t = t + 1
  end while
end
```

Proses dalam algoritma genetika diawali dengan **inisialisasi**, yaitu menciptakan individu-individu secara acak yang memiliki susunan gen (*chromosome*) tertentu. *Chromosome* ini mewakili solusi dari permasalahan yang akan dipecahkan. **Reproduksi** dilakukan untuk menghasilkan keturunan (*offspring*) dari individu-individu yang ada di populasi. **Evaluasi** digunakan untuk menghitung kebugaran (*fitness*) setiap *chromosome*. Semakin besar *fitness* maka semakin baik *chromosome* tersebut untuk dijadikan calon solusi. **Seleksi** dilakukan untuk memilih individu dari himpunan populasi dan *offspring* yang dipertahankan hidup pada generasi berikutnya. Fungsi probabilistis digunakan untuk memilih individu yang dipertahankan hidup. Individu yang lebih baik (mempunyai nilai kebugaran/*fitness* lebih besar) mempunyai peluang lebih besar untuk terpilih (Gen & Cheng 1997).

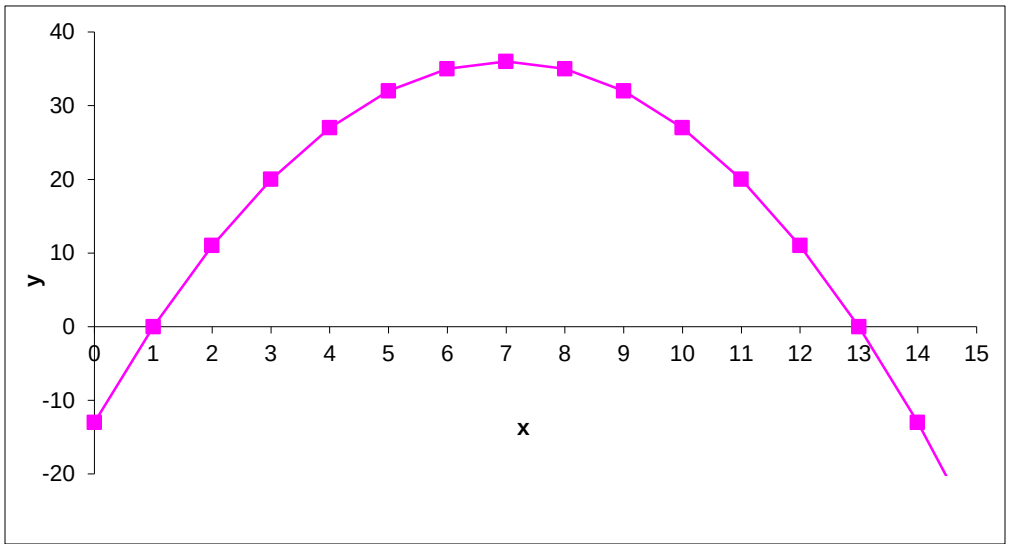
Setelah melewati sekian iterasi (generasi) akan didapatkan individu terbaik. Individu terbaik ini mempunyai susunan *chromosome* yang bisa dikonversi menjadi solusi yang terbaik (paling tidak mendekati optimum). Dari sini bisa disimpulkan bahwa algoritma genetika menghasilkan suatu solusi optimum dengan melakukan pencarian di antara sejumlah alternatif titik optimum berdasarkan fungsi probabilistic (Michalewicz 1996).

2.3. Studi Kasus: Maksimasi Fungsi Sederhana

Untuk menjelaskan siklus GAs maka diberikan contoh sederhana masalah maksimasi (mencari nilai maksimum) dari sebuah fungsi sebagai berikut:

$$\max \quad y = f(x) = -x^2 + 14x - 13, \qquad 0 \leq x \leq 15 \tag{2.1}$$

Grafik dari fungsi ini ditunjukkan pada Gambar 2.2. Nilai maksimum fungsi adalah *y*=36 pada *x*=7.



Gambar 2.2. Grafik fungsi contoh (2.1)

Dalam siklus perkembangan algoritma genetika mencari solusi (*chromosome*) 'terbaik' terdapat beberapa proses sebagai berikut:

2.3.1. Inisialisasi

Inisialisasi dilakukan untuk membangkitkan himpunan solusi baru secara acak/random yang terdiri atas sejumlah string chromosome dan ditempatkan pada penampungan

yang disebut populasi. Dalam tahap ini harus ditentukan ukuran populasi (*popSize*). Nilai ini menyatakan banyaknya individu/*chromosome* yang ditampung dalam populasi. Panjang setiap string *chromosome* (*stringLen*) dihitung berdasarkan presisi variabel solusi yang kita cari.

Misalkan kita tentukan *popSize*=4 dan kita gunakan representasi *chromosome* biner (bilangan basis 2). Nilai *x* ditentukan antara 0 sampai 15 dan bilangan biner dengan panjang 4 sudah dapat menjangkau nilai x (ingat $1111_2 = 15$). Jadi *stringLen*=4.

Misalkan kita dapatkan populasi inisial dan konversi *chromosome*-nya menjadi *x* sebagai berikut:

	chromosome	*x*	*y=f(x)*
P_1	[0 0 1 1]	3	20
P_2	[0 1 0 0]	4	27
P_3	[1 0 0 1]	9	32
P_4	[0 1 0 1]	5	32

2.3.2. Reproduksi

Reproduksi dilakukan untuk menghasilkan keturunan dari individu-individu yang ada di populasi. Himpunan keturunan ini ditempatkan dalam penampungan *offspring*. Dua operator genetika yang digunakan dalam proses ini adalah tukar silang (*crossover*) dan mutasi (*mutation*). Ada banyak metode *crossover* dan *mutation* yang telah dikembangkan oleh para peneliti dan biasanya bersifat spesifik terhadap masalah dan representasi *chromosome* yang digunakan.

Dalam tahap ini harus ditentukan tingkat *crossover* (*crossover rate / cr*). Nilai ini menyatakan rasio *offspring* yang dihasilkan proses *crossover* terhadap ukuran populasi sehingga akan dihasilkan *offspring* sebanyak *cr* x *popSize*.

Nilai tingkat mutasi (*mutation rate / mr*) juga harus ditentukan. Nilai ini menyatakan rasio *offspring* yang dihasilkan dari proses mutasi terhadap ukuran populasi sehingga akan dihasilkan *offspring* sebanyak *mr* x *popSize*.

Jika kita tentukan *cr*=0,5 maka ada 0,5×4=2 *offspring* yang dihasilkan dari proses *crossover*. Jika kita tentukan setiap *crossover* menghasilkan dua anak maka hanya ada satu kali operasi *crossover*. Misalkan P_1 dan P_3 terpilih sebagai *parent* maka akan kita dapatkan *offspring* C_1 dan C_2 sebagai berikut:

P_1 [**0 0 1 1**]
P_3 [**1 0 0 1**]

C_1 [**0 0 0 1**]
C_2 [**1 0 1 1**]

Setiap *offspring* mewarisi susunan gen (*chromosome*) dari induknya. Perhatikan dua bit pertama dari C_1 didapatkan dari P_1 dan sisanya dua bit terakhir dari P_3. C_2 mewarisi dua bit pertama dari P_3 dan sisanya dua bit terakhir dari P_1. Metode ini selanjutnya disebut *one-cut-point crossover*.

Jika kita tentukan *mr*=0,2 maka ada 0,2×4=0,8 (dibulatkan jadi 1) *offspring* yang dihasilkan dari proses mutasi. Misalkan P_4 terpilih sebagai *parent* maka akan kita dapatkan *offspring* ke-3 (C_3) sebagai berikut:

P_4 [0 1 0 **1**]
C_3 [0 1 0 **0**]

Perhatikan proses mutasi dilakukan hanya dengan memilih satu gen secara random kemudian mengubah nilainya.

Sekarang kita mempunyai 3 individu dalam penampungan *offspring* sebagai berikut:

	chromosome
C_1	[**0 0 0 1**]
C_2	[**1 0 1 1**]
C_3	[**0 1 0 1**]

2.3.3. Evaluasi

Evaluasi digunakan untuk menghitung kebugaran (*fitness*) setiap *chromosome*. Semakin besar *fitness* maka semakin baik *chromosome* tersebut untuk dijadikan calon solusi.

Karena sebuah *chromosome* selalu memiliki nilai *fitness* dan beberapa properti lain, maka dalam pembahasan berikutnya seringkali digunakan istilah 'individu'. Hal ini bisa

dianalogikan dengan seorang manusia sebagai individu. Dia memiliki tubuh beserta susunan gen pembentuknya (*chromosome*), nama, umur, alamat dan properti lainnya.

Dari proses inisialisasi dan reproduksi kita sekarang mempunyai kumpulan individu sebagai berikut:

	chromosome	*x*	*y=f(x)*	*fitness*
P_1	[0 0 1 1]	3	20	20
P_2	[0 1 0 0]	4	27	27
P_3	[1 0 0 1]	9	32	32
P_4	[0 1 0 1]	5	32	32
C_1	[0 0 0 1]	1	0	0
C_2	[1 0 1 1]	11	20	20
C_3	[0 1 0 0]	4	27	27

Karena masalah ini adalah pencarian nilai maksimum, maka nilai *fitness* untuk tiap individu bisa dihitung secara langsung *fitness=f(x)*.

2.3.4. Seleksi

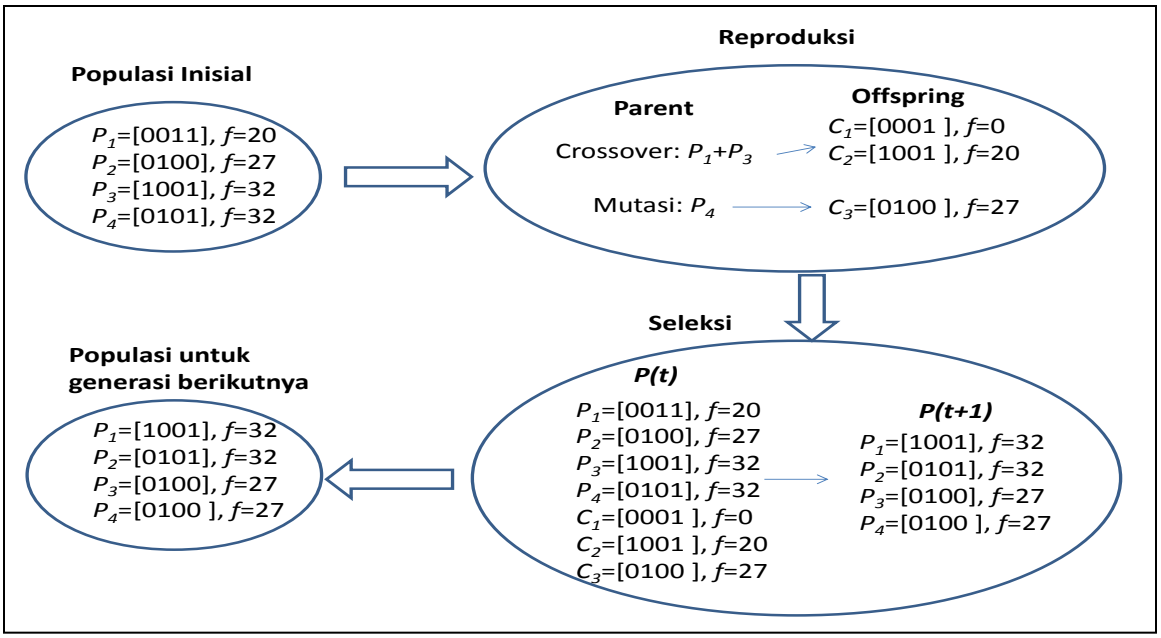
Seleksi dilakukan untuk memilih individu dari himpunan populasi dan *offspring* yang dipertahankan hidup pada generasi berikutnya. Semakin besar nilai *fitness* sebuah *chromosome* maka semakin besar peluangnya untuk terpilih. Hal ini dilakukan agar terbentuk generasi berikutnya yang lebih baik dari generasi sekarang. Metode seleksi yang sering digunakan adalah *roulette wheel*, *binary tournament*, dan *elitism*. Pembahasan metode-metode ini secara detail beserta *pseudo-code* nya ada pada subbab selanjutnya.

Misalkan kita gunakan *elitism selection*. Metode ini memilih *popSize* individu terbaik dari kumpulan individu di populasi (*parent*) dan *offspring*. Dengan cara ini maka P_3, P_4, P_2 dan C_3 terpilih. Sekarang kita mempunyai kumpulan individu yang bertahan hidup pada generasi berikutnya sebagai berikut:

asal pada *P(t)*	*P(t+1)*	chromosome	*x*	*y=f(x)*	*fitness*
P_3	P_1	[1 0 0 1]	9	32	32
P_4	P_2	[0 1 0 1]	5	32	32
P_2	P_3	[0 1 0 0]	4	27	27
C_3	P_4	[0 1 0 0]	4	27	27

Sampai tahap ini kita mempunyai P_1 (atau P_2) sebagai individu terbaik karena mempunyai nilai *fitness* terbesar.

Siklus proses 2 sampai proses 4 ini akan berlangsung berulang kali (sekian generasi) sampai tidak dihasilkan perbaikan keturunan, atau sampai kriteria optimum (*f(x)* maksimum) ditemukan (tidak ditemukan lagi individu dengan *fitness* yang lebih baik). Siklus lengkap dari contoh ini ditunjukkan pada Gambar 2.3. Penjelasan berbagai macam kriteria penghentian iterasi GAs diberikan pada Sub-Bab 2.5.



Gambar 2.3. Siklus Algoritma Genetika

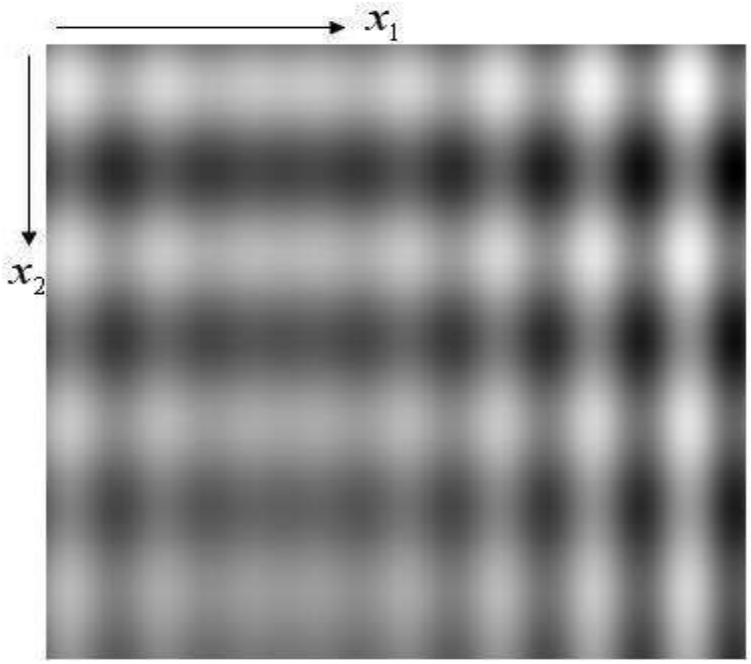
2.4. Studi Kasus: Maksimasi Fungsi dengan Presisi Tertentu

Untuk memperjelas uraian pada Sub-Bab 2.3 dan bagaimana menangani angka pecahan (desimal) serta penggunaan seleksi *roulette wheel* maka diberikan lagi contoh sederhana masalah maksimasi (mencari nilai maksimum) dari sebuah fungsi sebagai berikut:

$$\max \quad f(x_1, x_2) = 19 + x_1 \sin(x_1 \pi) + (10 - x_2) \sin(x_2 \pi) \qquad (2.2)$$

$$-5{,}0 \le x_1 \le 9{,}8, \qquad 0{,}0 \le x_2 \le 7{,}3$$

Plotting dua dimensi dari fungsi ini ditunjukkan pada Gambar 2.4. Warna putih menunjukkan nilai fungsi yang lebih besar. Perhatikan bahwa fungsi ini mempunyai banyak nilai maksimum lokal.



Gambar 2.4. Plotting 2D fungsi contoh (2.2)

2.4.1. Representasi Chromosome

Dalam kasus ini variabel keputusan (x_1 dan x_2) dikodekan dalam string *chromosome* biner. Panjang string *chromosome* tergantung pada presisi yang dibutuhkan. Misalkan domain variabel x_j adalah $[a_j,b_j]$ dan presisi yang dibutuhkan adalah *d* angka di belakang titik desimal, maka interval dari domain tiap variabel setidaknya mempunyai lebar (b_j - a_j)×10^d. Banyaknya bit yang dibutuhkan (m_j) untuk variabel x_j adalah (Gen & Cheng 1997):

$$\left(b_j - a_j\right) \times 10^d \leq 2^{m_j} - 1 \tag{2.3}$$

Konversi (*decoding*) dari substring biner menjadi bilangan real untuk variabel x_j adalah sebagai berikut:

$$x_j = a_j + decimal(substring)\frac{b_j - a_j}{2^{m_j} - 1} \tag{2.4}$$

decimal(substring) merepresentasikan nilai desimal dari substring bagi variabel keputusan x_j.

Contoh:

Misalkan untuk variabel x_1 dan x_2 kita tentukan mempunyai ketelitian 4 angka di belakang titik desimal, maka kebutuhan bit untuk kedua variabel tersebut adalah:

$$\big(9{,}8-(-5)\big)\times 10^4 \le 2^{m_1}-1$$

$$148.001 \le 2^{m_1} \quad \rightarrow \quad m_1 = 18$$

$$(7{,}3-0)\times 10^4 \le 2^{m_2}-1$$

$$73.001 \le 2^{m_2} \quad \rightarrow \quad m_2 = 17$$

Maka panjang string *chromosome* adalah $m_1 + m_2 = 35$. Misalkan sebuah string *chromosome* yang dibangkitkan secara random adalah sebagai berikut:

P_j	011010101010110001	00010010001001000
	18 bit	17 bit

Maka *chromosome* tersebut bisa diuraikan (decoding) sebagai berikut:

	angka biner	angka desimal
x_1	011010101010110001	109.233
x_2	00010010001001000	9.288

$$x_1 = -5 + 109.233 \times \frac{9{,}8-(-5)}{2^{18}-1} = 1{,}1670$$

$$x_2 = 0 + 9.288 \times \frac{7{,}3-(0)}{2^{17}-1} = 0{,}5173$$

2.4.2. Inisialisasi

Populasi inisial dibangkitkan secara random. Misalkan ditentukan *popSize*=10 maka akan dihasilkan populasi seperti contoh berikut:

	chromosome	x_1	x_2	$f(x_1,x_2)$
P_1	[00110000010000001010010111111111010]	-2,2104	4,3341	25,2705
P_2	[10011110111101111111110110100011111]	4,1904	7,0309	21,0716

P_3	[10101101000100010101010110010111001]	5,0055	2,4627	26,3991
P_4	[10000001100101010101010100011111001]	2,4915	2,4092	28,7747
P_5	[10010001001011001010110111000110010]	3,3929	5,2212	12,7376
P_6	[00011100000100010101010110001011101]	-3,3773	2,4575	23,3457
P_7	[10101101000110011100011010001011101]	5,0074	0,7466	25,4962
P_8	[00011100000100010101010110001011101]	-3,3773	2,4575	23,3457
P_9	[00110000011000001010010111111111010]	-2,2032	4,3341	25,2263
P_{10}	[10101101000110011100011010001011101]	5,0074	0,7466	25,4962
			Rata-Rata	23,7163

Nilai x_1, x_2, dan $f(x_1,x_2)$ didapatkan melalui proses decoding. Perhitungan fitness akan dijelaskan pada Sub-Bab 2.4.4. Individu terbaik pada populasi inisial ini adalah P_4 dengan *fitness*=$f(x_1,x_2)$= 28,7747.

2.4.3. Reproduksi

Crossover dilakukan dengan memilih dua induk (*parent*) secara acak dari populasi. Metode *crossover* yang digunakan adalah metode *one-cut-point*, yang secara acak memilih satu titik potong dan menukarkan bagian kanan dari tiap induk untuk menghasilkan *offspring*.

Misalkan yang terpilih sebagai induk adalah P_6 dan P_2. Titik potong (*cut point*) adalah titik 2. Maka akan dihasilkan dua *offspring* (C_1 dan C_2) sebagai berikut:

↓ titik potong

P_6 [00 01110000010001010101011000101110 1]
P_2 [10 01111011110111111111011010001111 1]

C_1 [00 01111011110111111111011010001111 1]
C_2 [10 01110000010001010101011000101110 1]

Jika kita tentukan *cr*=0,4 maka ada 0,4×10=4 *offspring* yang dihasilkan. Karena setiap *crossover* menghasilkan dua anak maka ada dua kali operasi *crossover*.

Misalkan yang terpilih sebagai induk pada *crossover* ke-2 adalah P_9 dan P_7. Titik potong (*cut point*) adalah titik 11. Maka akan dihasilkan dua *offspring* (C_3 dan C_4) sebagai berikut:

C_3 [00110000011110011100011010001011101]
C_4 [10101101000000001010010111111111010]

Mutasi dilakukan dengan memilih satu induk secara acak dari populasi. Metode mutasi yang digunakan adalah dengan memilih satu titik acak kemudian mengubah nilai gen pada titik tersebut.

Misalkan yang terpilih sebagai induk adalah P_9. Titik acak yang terpilih adalah 18. Maka akan dihasilkan *offspring* (C_5) sebagai berikut:

titik terpilih ↓

P_9 [0011000001100000**1**0100101111111110 10]

C_5 [0011000001100000**1**1001011111111110 10]

Anggap kita tentukan *mr*=0,2 maka ada 0,2×10=2 *offspring* yang dihasilkan dari proses mutasi. Misalkan yang terpilih sebagai induk pada mutasi ke-2 adalah P_1 dan titik acak yang terpilih adalah 19 maka akan dihasilkan *offspring* (C_6) sebagai berikut:

C_6 [00110000010000001000010111111111010]

Keseluruhan *offspring* yang dihasilkan dari proses reproduksi (*crossover* dan mutasi) adalah sebagai berikut:

	chromosome
C_1	[00011110111101111111110110100011111]
C_2	[10011100000100010101010110001011101]
C_3	[00110000011110011100011010001011101]
C_4	[10101101000000001010010111111111010]
C_5	[00110000011000001110010111111111010]
C_6	[00110000010000001000010111111111010]

Perhatikan bahwa sekarang kita mempunyai 16 individu (10 dari populasi awal ditambah 6 *offspring*).

2.4.4. Evaluasi

Evaluasi dilakukan terhadap keseluruhan 16 individu dengan cara:

- Mengubah/memetakan setiap string biner menjadi variabel x_1 dan x_2.
- Menghitung nilai fungsi obyektive $f(x_1,x_2)$.

- Konversi nilai *f(x_1,x_2)* menjadi nilai *fitness*. Karena masalah ini adalah pencarian nilai maksimum, maka nilai *fitness* untuk tiap individu bisa dihitung secara langsung sebagai berikut:

$$fitness = f(x_1, x_2) \tag{2.5}$$

Untuk masalah pencarian nilai minimum maka bisa digunakan rumusan (2.6) atau (2.7). *C* pada (2.6) merupakan nilai konstan yang harus ditetapkan sebelumnya. Penggunaan (2.7) harus dilakukan secara hati-hati untuk memastikan tidak terjadi pembagian dengan nol.

$$fitness = C - f(X) \tag{2.6}$$

$$fitness = \frac{1}{f(X)} \tag{2.7}$$

2.4.5. Seleksi

Seleksi dilakukan untuk memilih 10 dari 16 individu yang dipertahankan hidup pada generasi berikutnya. Metode yang digunakan adalah *roulette wheel*. Pendekatan ini dilakukan dengan menghitung nilai probabilitas seleksi (*prob*) tiap individu berdasarkan nilai fitnessnya. Dari nilai *prob* ini bisa dihitung probabilitas kumulatif (*probCum*) yang digunakan pada proses seleksi tiap individu.

Langkah-langkah membentuk *roulette wheel* berdasarkan probabilitas kumulatif adalah:

- Misalkan *fitness(P_k)* adalah nilai *fitness* individu ke-*k*. Maka bisa dihitung total *fitness* sebagai berikut:

$$totalFitness = \sum_{k=1}^{popSize} fitness(P_k)$$

Perhatikan bahwa nilai *popSize* sekarang adalah 16.

- Hitung nilai probabilitas seleksi (*prob*) tiap individu:

$$prob_k = \frac{fitness(P_k)}{totalFitness}, \qquad k = 1,2,\dots,popSize$$

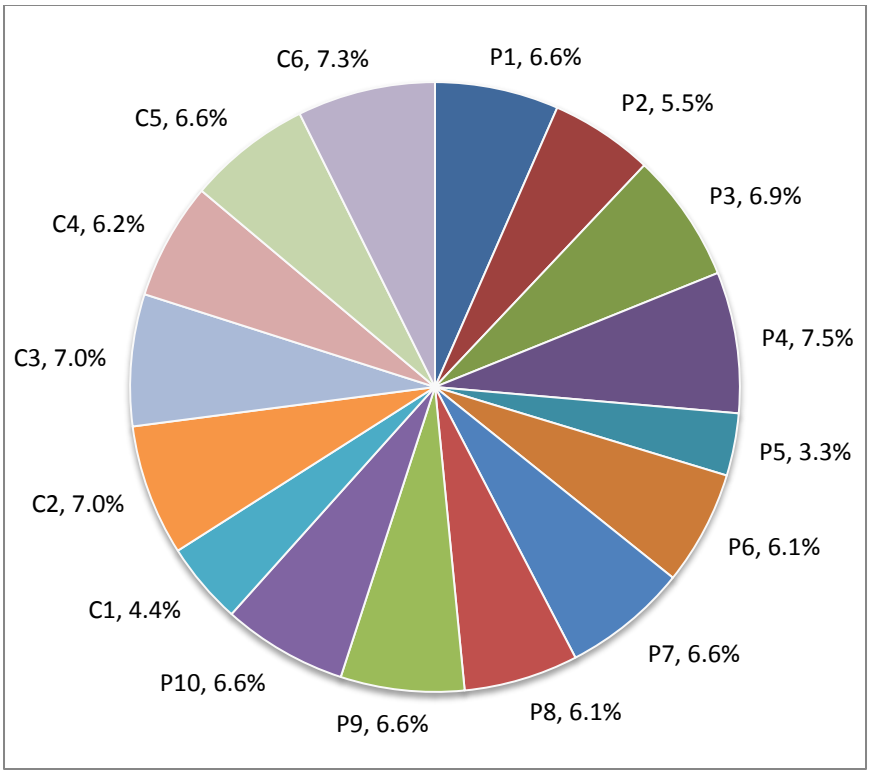
- Hitung nilai probabilitas kumulatif (*probCum*) tiap individu:

$$probCum_k = \sum_{j=1}^{k} prob_j\,, \qquad k = 1,2,\dots,popSize$$

Dari proses evaluasi dan perhitungan probabilitas kumulatif, kita sekarang mempunyai kumpulan individu sebagai berikut:

	x_1	x_2	$f(x_1,x_2)$	fitness	prob	probCum
P_1	-2,2104	4,3341	25,2705	25,2705	0,0657	0,0657
P_2	4,1904	7,0309	21,0716	21,0716	0,0548	0,1204
P_3	5,0055	2,4627	26,3991	26,3991	0,0686	0,1890
P_4	2,4915	2,4092	28,7747	28,7747	0,0748	0,2638
P_5	3,3929	5,2212	12,7376	12,7376	0,0331	0,2969
P_6	-3,3773	2,4575	23,3457	23,3457	0,0607	0,3576
P_7	5,0074	0,7466	25,4962	25,4962	0,0663	0,4238
P_8	-3,3773	2,4575	23,3457	23,3457	0,0607	0,4845
P_9	-2,2032	4,3341	25,2263	25,2263	0,0656	0,5500
P_{10}	5,0074	0,7466	25,4962	25,4962	0,0663	0,6163
C_1	-3,2097	7,0309	16,7478	16,7478	0,0435	0,6598
C_2	4,0227	2,4575	26,7618	26,7618	0,0695	0,7293
C_3	-2,1975	0,7466	26,8905	26,8905	0,0699	0,7992
C_4	5,0017	4,3341	23,8865	23,8865	0,0621	0,8613
C_5	-2,2031	4,3341	25,2259	25,2259	0,0655	0,9268
C_6	-2,2104	0,684	28,1587	28,1587	0,0732	1,0000
Total				384,8348		

Dengan nilai probabilitas seleksi (*prob*) tiap individu maka kita mempunyai *roulette wheel* sebagai berikut:



Gambar 2.5. Roulette wheel peluang terpilihnya setiap individu

Perhatikan ukuran tiap potongan pie pada Gambar 2.5 adalah sesuai dengan probabilitas (*prob* x 100%) terpilihnya setiap individu.

Sekarang waktunya memutar *roulette wheel* untuk memilih setiap individu dengan langkah-langkah berikut:

- Bangkitkan *r* bilangan random pada range [0,1].
- Pilih/cek nilai *k* mulai dari 1 sampai *popSize* sehingga $r \leq probCum_k$.

  Misal *r*=0,5210 maka *k*=9, artinya individu yang ke-9 terpilih. Hasil selengkapnya bisa dilihat sebagai berikut:

P(t+1)	random	k	individu terpilih	chromosome	fitness
P_1	0,5210	9	P_9	[0011000001100000101001011111111010]	25,2263
P_2	0,4307	8	P_8	[0001110000010001010101011000101101]	23,3457
P_3	0,5297	9	P_9	[0011000001100000101001011111111010]	25,2263
P_4	0,9050	15	C_5	[0011000001100000111001011111111010]	25,2259
P_5	0,0745	2	P_2	[1001111011110111111111101101000111111]	21,0716
P_6	0,0900	2	P_2	[1001111011110111111111101101000111111]	21,0716
P_7	0,7803	13	C_3	[0011000001111001110001101000101101]	26,8905

P_8	0,4032	7	P_7	[10101101000110011100011010001011101]	25,4962
P_9	0,1680	3	P_3	[10101101000100010101010110010111001]	26,3991
P_{10}	0,4594	8	P_8	[00011100000100010101010110001011101]	23,3457

Sekarang kita telah menyelesaikan satu iterasi (generasi) dengan individu terbaik adalah P_7 dengan fitness=26,8905 yang bisa diuraikan (*decoding*) sebagai berikut:

x_1 = -2,1975

x_2 = 0,7466

$f(x_1,x_2)$ = 26,8905

Perhatikan individu terbaik pada generasi ke-1 (*fitness*=26,8905) ini lebih buruk dari pada individu terbaik pada populasi inisial (*fitness*=28,7747). Ingat dengan mekanisme *roulette wheel* maka individu dengan fitness lebih besar akan mempunyai peluang lebih besar untuk terpilih. Karena berupa peluang maka tidak menjamin bahwa individu terbaik akan selalu terpilih untuk masuk pada generasi berikutnya. Pada implementasi program komputer, individu terbaik ini disimpan pada variabel tersendiri untuk menghindari kemungkinan tereliminasi pada proses seleksi.

Program uji telah dijalankan sampai generasi ke-1000 dan hasil terbaik didapatkan pada generasi ke-322 sebagai berikut:

chromosome : [11101001110000011100010000110010110]

x_1 = 8,5141

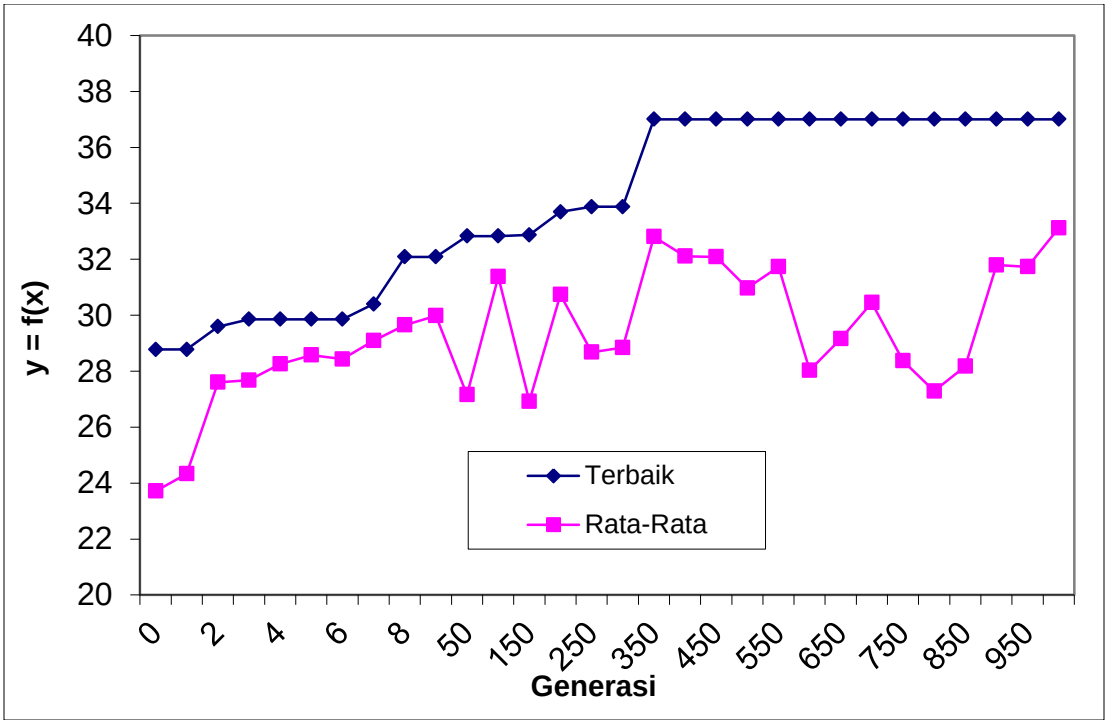
x_2 = 0,4789

$f(x_1,x_2)$ = 37,0059

Hasil untuk beberapa generasi telah ditabelkan dan dibuat grafiknya pada Gambar 2.6 untuk menunjukkan solusi yang dicapai oleh GA sampai generasi ke-1000. Perhatikan bahwa nilai rata-rata fungsi obyektif bersifat fluktuatif tapi cenderung naik sepanjang generasi.

generasi	$f(x_1,x_2)$	
	terbaik	rata-rata
0	28,7747	23,7164
1	28,7747	24,3299
2	29,5915	27,6008
3	29,8536	27,6772
4	29,8536	28,2518
5	29,8536	28,5771
6	29,8536	28,4292
7	30,4004	29,0905
8	32,0874	29,6504
9	32,0874	29,9816
50	32,8296	27,1568
100	32,8296	31,3788
150	32,8699	26,9207
200	33,6935	30,7364
250	33,8751	28,6786

generasi	$f(x_1,x_2)$	
	terbaik	rata-rata
300	33,8751	28,8446
350	37,0059	32,8052
400	37,0059	32,1089
450	37,0059	32,0864
500	37,0059	30,9705
550	37,0059	31,7303
600	37,0059	28,0305
650	37,0059	29,1582
700	37,0059	30,4500
750	37,0059	28,3685
800	37,0059	27,2866
850	37,0059	28,1736
900	37,0059	31,7932
950	37,0059	31,7336
1000	37,0059	33,1172



Gambar 2.6. Solusi GA tiap generasi untuk fungsi uji (2.2)

2.5. Kondisi Berhenti (*Termination Condition*)

Iterasi GAs diulang terus sampai kondisi berhenti tercapai. Beberapa kriteria bisa dipakai untuk hal ini sebagai berikut:

1. Iterasi berhenti sampai generasi *n*. Nilai *n* ditentukan sebelumnya berdasarkan beberapa eksperimen pendahuluan. Semakin tinggi ukuran dan kompleksitas masalah maka nilai *n* semakin besar. Nilai *n* ditentukan sedemikian rupa sehingga konvergensi populasi tercapai dan akan sulit didapatkan solusi yang lebih baik setelah *n* iterasi (Yogeswaran, Ponnambalam & Tiwari 2009).
2. Iterasi berhenti setelah *n* generasi berurutan tidak dijumpai solusi yang lebih baik (Mahmudy, Marian & Luong 2012b). Kondisi ini menunjukkan bahwa GAs sulit mendapatkan solusi yang lebih baik dan penambahan iterasi hanya membuang waktu.
3. Iterasi berhenti setelah *t* satuan waktu tercapai. Ini biasa digunakan jika diinginkan untuk membandingkan performa dari beberapa algoritma (Mahmudy 2014a).

Dalam implementasi praktis, kombinasi kondisi (1) dan (2) bisa dipakai (Mahmudy, Marian & Luong 2013b).

2.6. Rangkuman

Pada bab ini telah dibahas struktur GAs beserta siklusnya. GAs merupakan sub-kelas paling popular dari algoritma evolusi. Kelebihan GAs sebagai metode optimasi dirangkumkan dari berbagai sumber literatur.

Studi kasus maksimasi fungsi sederhana diberikan untuk memperjelas beberapa tahapan dalam penyelesaian masalah menggunakan GAs yang meliputi inisialisasi chromosome, proses reproduksi menggunakan *crossover* dan mutasi, evaluasi chromosome menggunakan fungsi fitness, dan proses seleksi menggunakan metode *elitism*. Satu metode crossover untuk representasi biner, *one-cut-point crossover*, dikenalkan pada Sub-Bab 2.3.2.

Satu kasus lagi diberikan pada Sub-Bab 2.4 untuk menjelaskan bagaimana menangani angka pecahan (desimal) serta penggunaan seleksi *roulette wheel*.

2.7. Latihan

Untuk memperjelas pemahaman anda, kerjakanlah latihan berikut sebisa mungkin tanpa melihat materi pada buku!

1. Sebutkan semua proses utama dalam siklus algoritma genetika?
2. Jelaskan keunggulan algoritma genetika sebagai algoritma yang berbasis populasi!
3. Jelaskan apa yang dimaksud dengan algoritma genetika bersifat *ergodic*?
4. Iterasi GAs diulang terus sampai kondisi berhenti tercapai. Jelaskan beberapa kriteria untuk hal ini?
5. Misalkan P_1 dan P_2 adalah *parent* untuk proses crossover. Tentukan offspring yang terbentuk jika dilakukan *one-cut-point crossover* pada titik ke-5.

   P_1 **[0 0 1 1 0 0 1]**
   P_2 **[1 0 0 1 1 1 1]**

6. Misalkan P adalah *parent* untuk proses mutasi. Tentukan offspring yang terbentuk jika dilakukan mutasi pada titik ke-5.

   P **[0 0 1 1 0 0 1]**

7. Untuk masalah maksimasi (mencari nilai maksimum) dari sebuah fungsi sebagai berikut

$$\max \; y = f(x) = -\frac{x^2}{2} + 4x + 40, \qquad 0 \le x \le 15$$

   Lengkapi tabel berikut:

	chromosome	*x*	*y=f(x)*	*fitness*
P_1	[0 0 0 1]			
P_2	[1 1 0 0]			
P_3	[1 0 1 1]			
P_4	[1 1 1 1]			

8. Untuk fungsi uji (2.2) maksimasi fungsi dengan presisi tertentu, lengkapi tabel berikut:

chromosome	x_1	x_2	$f(x_1,x_2)$

P_1	[10101101000110011100011010001011101]
P_2	[00011100000100010101010110001011101]
P_3	[00110000011000001010010111111111010]
P_4	[10101101000110011100011010001011101]

9. Pada table berikut *P* menunjukkan *parent* dan *C* menunjukkan *offspring*. Untuk seleksi *roulette wheel*, lengkapi kolom untuk probabilitas dan probabilitas kumulatif!

	fitness	prob	probCum
P_1	25,2705		
P_2	21,0716		
P_3	26,3991		
P_4	28,7747		
C_1	12,7376		
C_2	23,3457		

10. Untuk soal no. 9, tentukan empat individu yang terpilih jika diberikan angka random 0,5342, 0,2189, 0,1987, dan 0,8652!
11. Untuk fungsi uji (2.2), jika variabel x_1 dan x_2 kita tentukan mempunyai ketelitian 5 angka di belakang titik desimal, hitung kebutuhan bit untuk kedua variabel tersebut!

BAB 3

Algoritma Genetika dengan Pengkodean Real (*Real-Coded GA/RCGA*)

Kelemahan algoritma genetika dengan pengkodean biner jika digunakan pada optimasi fungsi adalah tidak bisa menjangkau beberapa titik solusi jika range solusi berada dalam daerah kontiyu. Selain itu, pada optimasi fungsi yang kompleks dan membutuhkan banyak generasi, operasi transformasi biner ke bilangan desimal (real) dan sebaliknya sangat menyita waktu. Pengkodean real (*real-coded genetic algorithms, RCGA*) bisa menyelesaikan masalah ini (Herrera, Lozano & Verdegay 1998). Dengan fungsi yang sama pada Sub-Bab 2.4 akan diberikan contoh bagaimana *RCGA* bekerja.

3.1. Siklus RCGA

Sub-bab ini membahas secara detail siklus RCGA.

3.1.1. Representasi Chromosome

Dalam kasus ini variabel keputusan (x_1 dan x_2) langsung menjadi gen string *chromosome*, sehingga panjang string *chromosome* adalah 2.

3.1.2. Inisialisasi

Populasi inisial dibangkitkan secara random. Misalkan ditentukan *popSize*=10 maka akan dihasilkan populasi sebagai berikut:

	chromosome		
	x_1	x_2	$f(x_1,x_2)$
P_1	1,4898	2,0944	19,8206
P_2	8,4917	2,5754	34,7058
P_3	1,4054	6,3035	20,6707
P_4	5,8114	5,0779	14,5624
P_5	-1,8461	1,7097	11,5858
P_6	4,0206	4,4355	24,7106
P_7	-0,1634	2,974	19,653

P_8	5,2742	0,7183	22,1813
P_9	9,4374	6,6919	12,4694
P_{10}	-4,5575	0,1679	28,4324

3.1.3. Reproduksi

Crossover dilakukan dengan memilih dua induk (*parent*) secara acak dari populasi. Metode *crossover* yang digunakan adalah *extended intermediate crossover* (Muhlenbein & Schlierkamp-Voosen 1993) yang menghasilkan *offspring* dari kombinasi nilai dua induk. Misalkan P_1 dan P_2 adalah dua kromosom yang telah diseleksi untuk melakukan crossover, maka offspring C_1 dan C_2 bisa dibangkitkan sebagai berikut:

$$C_1 = P_1 + \alpha\ (P_2 - P_1)$$

$$C_2 = P_2 + \alpha\ (P_1 - P_2)$$

α dipilih secara acak pada interval [-0,25, 1,25].

Misalkan yang terpilih sebagai induk adalah P_4 dan P_9, α=[0,1104, 1,2336] maka akan dihasilkan dua *offspring* (C_1 dan C_2) sebagai berikut:

C_1 : x_1= 5,8114 + 0,1104 (9,4374-5,8114) = 6,2118

x_2= 5,0779 + 1,2336 (6,6919-5,0779) = 7,0690

C_2 : x_1= 9,4374 + 0,1104 (5,8114-9,4374) = 9,0370

x_2= 6,6919 + 1,2336 (5,0779-6,6919) = 4,7008

Jika kita tentukan pc=0,4 maka ada 0,4×10=4 *offspring* yang dihasilkan. Karena setiap *crossover* menghasilkan dua anak maka ada dua kali operasi *crossover*. Anggap dua offspring berikutnya adalah C_3 dan C_4.

Mutasi dilakukan dengan memilih satu induk secara acak dari populasi. Metode mutasi yang digunakan adalah *random mutation* yang dilakukan dengan menambah atau mengurangi nilai gen terpilih dengan bilangan random yang kecil. Misalkan domain

variabel x_j adalah $[min_j, max_j]$ dan offspring yang dihasilkan adalah $C=[x'_1..x'_n]$, maka nilai gen *offspring* bisa dibangkitkan sebagai berikut:

$$x'_i = x'_i + r\,(max_i - min_j)$$

range r misalkan [-0,1, 0,1].

Misalkan yang terpilih sebagai induk adalah P_2, gen yang terpilih nomer 2 (x_2) dan r=-0,0584. Maka akan dihasilkan *offspring* (C_5) sebagai berikut:

C_5 : x_1= 8,4917 (tetap)

x_2= 2,5754 – 0,0584 (7,3-0,0) = 2,1491

Anggap kita tentukan pm=0,2 maka ada 0,2×10=2 *offspring* yang dihasilkan dari proses mutasi. Anggap *offspring* berikutnya adalah C_6.

Keseluruhan *offspring* yang dihasilkan dari proses reproduksi (*crossover* dan mutasi) adalah sebagai berikut:

	chromosome		$f(x_1,x_2)$
	x_1	x_2	
C_1	6,2118	7,0690	22,2048
C_2	9,0370	4,7008	22,2313
C_3	7,1636	0,0000	15,4774
C_4	7,5479	7,3000	9,3531
C_5	8,4917	2,1494	31,0389
C_6	-1,1238	1,7097	12,0177

Perhatikan bahwa sekarang kita mempunyai 16 individu (10 dari populasi mula-mula ditambah 6 *offspring*)

3.1.4. Seleksi

Seleksi dilakukan untuk memilih 10 dari 16 individu yang dipertahankan hidup pada generasi berikutnya. Metode yang digunakan adalah *tournament selection*. Pendekatan ini dilakukan dengan mengambil secara acak sejumlah kecil individu (biasanya 2, disebut *binary tournament selection*) dari penampungan populasi dan *offspring*. Satu individu dengan nilai fitness lebih besar akan terpilih untuk masuk populasi berikutnya. Langkah

ini diulangi sampai terpenuhi *popSize* individu terpilih. Pseudo-code *binary tournament selection* disajikan pada Gambar 3.1 sebagai berikut:

```
PROCEDURE BinaryTournamentSelection
Input:
     POP:       himpunan individu pada populasi
     pop_size:  ukuran populasi
     OS:        himpunan individu anak (offspring) hasil reproduksi
                menggunakan crossover and mutasi
     ns:        banyaknya offspring

Output:
     POP:       himpunan individu pada populasi setelah proses
                seleksi selesai

FOR i=1 TO ns DO
   /* pilih satu individu pada POP secara acak */
   p = Random (1, pop_size)
   IF Fitness(OS_i)> Fitness(POP_p) THEN
      POP_p ← OS_i
   END IF
END FOR

END PROCEDURE
```

Gambar 3.1. Pseudo-code binary tournament selection

Selengkapnya hasil seleksi ini adalah:

P(t+1)	individu pertama	individu kedua	individu terpilih	fitness
P_1	P_4	P_9	P_4	14,5624
P_2	P_1	P_{10}	P_{10}	28,4324
P_3	P_1	C_{11}	C_{11}	22,2048
P_4	C_{13}	C_{16}	C_{13}	15,4774
P_5	C_{13}	P_9	C_{13}	15,4774
P_6	P_4	P_3	P_3	20,6707
P_7	P_1	C_{15}	C_{15}	31,0389
P_8	P_7	C_{13}	P_7	19,6530
P_9	P_8	P_6	P_6	24,7106
P_{10}	P_4	C_{11}	C_{11}	22,2048

Program uji telah dijalankan sampai generasi ke-1000 dan hasil terbaik didapatkan pada generasi ke-847 sebagai berikut:

x_1 = 8,5113

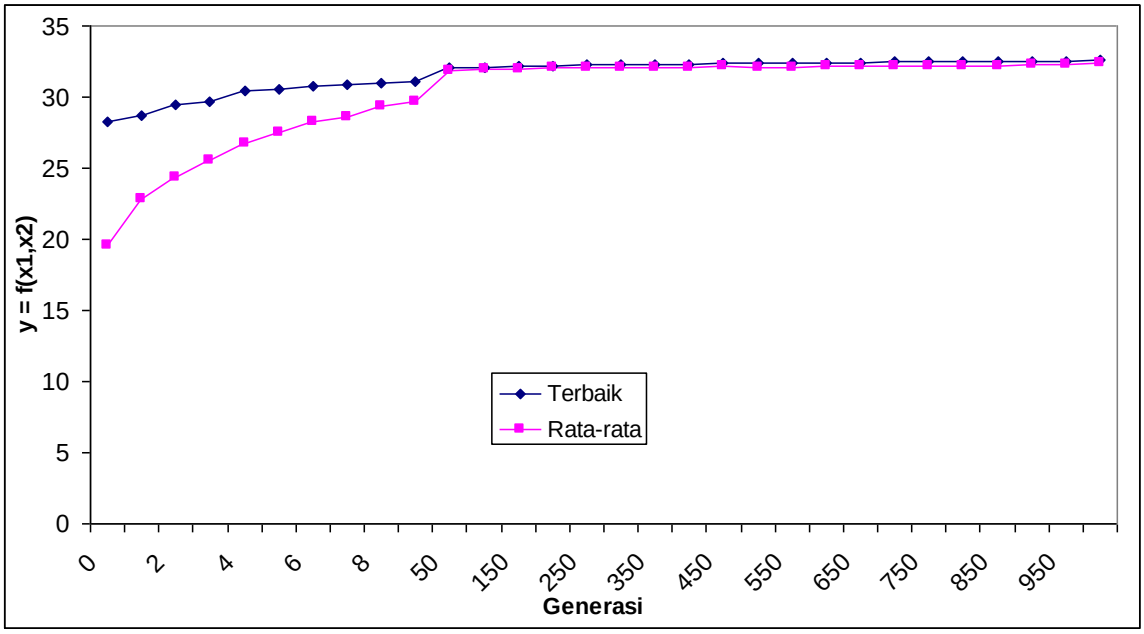
x_2 = 2,4865

$f(x_1,x_2)$ = 35,0127

Hasil untuk beberapa generasi telah ditabelkan dan dibuat grafiknya pada Gambar 3.2 untuk menunjukkan solusi yang dicapai oleh GA sampai generasi ke-1000.

generasi	$f(x_1,x_2)$	
	terbaik	rata-rata
0	28,3061	19,5587
1	28,6909	22,8057
2	29,4916	24,3434
3	29,6801	25,4945
4	30,4030	26,6921
5	30,5790	27,5221
6	30,7323	28,2436
7	30,8796	28,6316
8	30,9930	29,3526
9	31,0899	29,6470
50	32,0672	31,8526
100	32,1134	31,9385
150	32,1303	31,9970
200	32,1990	32,0365
250	32,2386	32,0225

generasi	$f(x_1,x_2)$	
	terbaik	rata-rata
300	32,2609	32,0358
350	32,2610	32,0586
400	32,2818	32,0519
450	32,3695	32,1262
500	32,3695	32,0759
550	32,3696	32,0937
600	32,3757	32,1531
650	32,4447	32,1509
700	32,4711	32,2167
750	32,4711	32,2074
800	32,4712	32,1594
850	32,4712	32,2035
900	32,5104	32,2639
950	32,5508	32,2895
1000	32,5631	32,3744



Gambar 3.2. Solusi RCGA pada tiap generasi

Perhatikan ada masalah serius di sini yaitu hasil yang dicapai tidak sebaik representasi biner karena menggunakan seleksi *tournament* pada pengkodean real ternyata menyebabkan terjadinya konvergensi dini. Mulai generasi ke-50 hampir semua individu bernilai sama, perhatikan kondisi populasi pada generasi ke-1000:

	chromosome		$f(x_1,x_2)$
	x_1	x_2	
P_1	8,5113	2,4865	35,0127
P_2	8,5113	2,4865	35,0127
P_3	8,5113	2,4865	35,0127
P_4	8,5113	2,4865	35,0127
P_5	8,5113	2,4865	35,0127
P_6	8,5113	2,7325	32,9188
P_7	8,5113	2,4865	35,0127
P_8	9,6026	2,4865	17,3987
P_9	8,5113	2,4865	35,0127
P_{10}	8,5113	2,4865	35,0127

Pada kondisi seperti ini maka proses reproduksi juga akan menghasilkan *offspring* yang hampir sama dengan induknya sehingga eksplorasi solusi tidak berjalan baik. Bagaimana untuk mengatasi kondisi seperti ini akan dibahas mendetail pada subbab berikutnya.

3.2. Alternatif Operator Reproduksi pada Pengkodean Real

Metode *one-cut-point crossover* yang digunakan pada pengkodean chromosome biner bisa dengan mudah diterapkan pada pengkodean real. Misalkan pada kasus optimasi fungsi dengan 6 variabel keputusan, P_1 dan P_1 adalah *parent*, titik potong pada posisi ke-4, maka akan kita dapatkan *offspring* C_1 dan C_2 sebagai berikut:

P_1 [**0,078 9,231 7,629 3,517 3,619 1,498**]
P_2 [**1,903 8,729 2,578 4,529 0,592 2,337**]

C_1 [**0,078 9,231 7,629 3,517 0,592 2,337**]
C_2 [**1,903 8,729 2,578 4,529 3,619 1,498**]

Metode *one-cut-point crossover* ini akan memberikan hasil yang memuaskan jika dikombinasikan dengan operator crossover yang dikhususkan pada pengkodean real. Kombinasi dilakukan dengan memilih satu metode crossover secara acak setiap kali proses reproduksi dilakukan (Mahmudy, Marian & Luong 2012a, 2012b).

3.3. Alternatif Operator Seleksi

Metode seleksi *roulette wheel* telah dibahas dalam bab sebelumnya. Dalam bab ini telah dikenalkan metode seleksi *binary tournament*. Metode seleksi lain yang sering digunakan dalam penelitian adalah *elitism selection* dan *replacement selection*.

3.3.1. Elitism Selection

Metode seleksi *elitism* bekerja dengan mengumpulkan semua individu dalam populasi (*parent*) dan *offspring* dalam satu penampungan. *popSize* individu terbaik dalam penampungan ini akan lolos untuk masuk dalam generasi selanjutnya. Metode seleksi ini menjamin individu yang terbaik akan selalu lolos. Pseudo-code *elitism selection* disajikan pada Gambar 3.3 sebagai berikut:

```
PROCEDURE ElitismSelection
Input:
     POP:       himpunan individu pada populasi
     pop_size:  ukuran populasi
     OS:        himpunan individu anak (offspring) hasil reproduksi
                menggunakan crossover and mutasi

Output:
     POP:       himpunan individu pada populasi setelah proses
                seleksi selesai

/* gabungkan individu pada POP dan OS ke dalam TEMP */
TEMP ← Merge (POP,OS)
/* urutkan individu berdasarkan fitness secara ascending */
OrderAscending (Temp)
/* copy pop_size individu terbaik ke POP */
POP ← CopyBest (Temp, pop_size)

END PROCEDURE
```

Gambar 3.3. Pseudo-code *elitism selection*

Misalkan terdapat himpunan individu dalam populasi dengan *popSize*=5 sebagai berikut:

individu	fitness
P_1	10
P_2	8
P_3	4
P_4	7

P_5	6

Terdapat juga himpunan *offspring* sebagai berikut:

individu	fitness
C_1	3
C_2	8
C_3	5

Maka akan didapatkan himpunan individu yang lolos ke generasi berikutnya sebagai berikut:

P(t+1)	asal *P(t)*	fitness
P_1	P_1	10
P_2	P_2	8
P_3	C_2	8
P_4	P_4	7
P_5	P_5	6

Salah satu kelemahan dari *elitism selection* adalah tidak memberikan kesempatan kepada individu dengan nilai fitness rendah untuk bereproduksi. Dalam beberapa kasus, solusi optimum justru bisa dicapai dari hasil reproduksi individu dengan nilai fitness rendah. Penggunaan *random injection* yang akan dibahas di sub-bab berikutnya akan memperkuat kemampuan GAs yang menggunakan *elitism selection*.

3.3.2. Replacement Selection

Metode seleksi *replacement* mempunyai dua aturan sebagai berikut (Mahmudy, Marian & Luong 2013d, 2013e):

- Offspring yang diproduksi melalui proses mutasi menggantikan induknya jika mempunyai nilai fitness yang lebih baik.
- Offspring yang diproduksi melalui proses *crossover* (menggunakan dua induk) akan menggantikan induk yang terlemah jika mempunyai nilai fitness yang lebih baik daripada induk yang terlemah tersebut.

Metode seleksi *replacement* ini menjamin individu yang terbaik akan selalu lolos. Tetapi properti ini tidak menutup peluang individu dengan nilai fitness rendah untuk lolos ke generasi berikutnya. Hal ini merupakan keunggulan dari *replacement selection* karena

seperti telah diuraikan pada modul sebelumnya, solusi optimum mungkin didapatkan dari hasil reproduksi individu-individu dengan nilai fitness rendah.

Pseudo-code *replacement selection* disajikan pada Gambar 3.4 sebagai berikut:

```
PROCEDURE ReplacementSelection
Input:
     POP:       himpunan individu pada populasi
     OS:        himpunan individu anak (offspring) hasil reproduksi
                menggunakan crossover and mutasi
     ns:        banyaknya offspring
     P:         list dari indeks parent individu dalam offspring

Output:
     POP:       himpunan individu pada populasi setelah proses
                seleksi selesai

FOR i=1 TO ns DO
   /* get index of parent */
   p = P_i
   IF Fitness(OS_i)> Fitness(POP_p) THEN
      POP_p ← OS_i
   END IF
END PROCEDURE
```

Gambar 3.4. Pseudo-code *replacement selection*

Misalkan terdapat himpunan individu dalam populasi dengan *popSize*=5 sebagai berikut:

individu	fitness
P_1	10
P_2	8
P_3	4
P_4	7
P_5	6

Terdapat juga himpunan *offspring* sebagai berikut:

individu	parent	fitness
C_1	P_2 dan P_3	3
C_2	P_4 dan P_5	8
C_3	P_3	5

Maka akan didapatkan himpunan individu yang lolos ke generasi berikutnya sebagai berikut:

P(t+1)	asal *P(t)*	fitness
P_1	P_1	10
P_2	P_2	8
P_3	C_3	5
P_4	P_4	7
P_5	C_2	8

3.4. Diskusi: Nilai Parameter Algoritma Genetika

Menentukan nilai parameter yang tepat untuk algoritma genetika bukanlah pekerjaan mudah. Jika nilai parameter ukuran populasi (*popSize*), *crossover rate* (*cr*) dan *mutation rate* (*mr*) semakin besar maka akan meningkatkan kemampuan eksplorasi algoritma genetika untuk mencari solusi terbaik. Tetapi hal ini akan sangat membebani waktu komputasi (proses berlangsung lama) karena bisa jadi algoritma genetika akan mengeksplorasi area yang tidak mempunyai nilai optimum.

Tidak ada metode pasti untuk menentukan nilai parameter GAs. Kombinasi nilai yang tepat untuk parameter tersebut sangat dipengaruhi oleh permasalahan yang akan diselesaikan. Dalam penelitian optimasi menggunakan algoritma genetika, serangkaian pengujian pendahuluan diperlukan untuk mendapatkan kombinasi nilai parameter yang sesuai (Mahmudy, Marian & Luong 2014). Ukuran populasi (*popSize*) antara 30 sampai 50, *pc* antara 0,3 sampai 0,8, dan *pm* antara 0,1 sampai 0,3 biasanya sudah memadai untuk pengujian awal.

3.5. Diskusi: Mekanisme Sampling Proses Seleksi

Pada proses seleksi terdapat mekanisme sampling untuk memilih individu yang dipertahankan hidup. Ada tiga kategori metode dasar untuk melakukan sampling, yaitu (Gen & Cheng 1997):

- *Stochastic sampling*
- *Deterministic sampling*
- *Mixed sampling*

Stochastic sampling memilih menggunakan angka random dan berdasarkan nilai probabilitas. *Roulette wheel selection* merupakan contoh kategori ini, semakin besar nilai *fitness* sebuah individu maka semakin besar juga peluangnya untuk terpilih.

Deterministic sampling bekerja dengan aturan tetap, misalkan mengurutkan kumpulan individu (*parent+offspring*) berdasarkan nilai *fitness*-nya kemudian mengambil sejumlah individu dengan nilai *fitness* terbaik (sesuai dengan *popSize*). *Elitism selection* termasuk dalam kategori ini.

Mixed sampling merupakan strategi campuran dari stochastic sampling dan deterministic sampling. *Tournament selection* merupakan contoh kategori ini dengan memilih secara random 2 atau lebih individu kemudian mengambil satu yang terbaik.

Deterministic sampling menjamin individu terbaik akan selalu dipertahankan hidup. *Roulette wheel selection* dan *tournament selection* yang kita gunakan sebelumnya tidak menjamin individu terbaik akan selalu terpilih. Dalam implementasi program disediakan variabel tersendiri untuk menyimpan nilai individu terbaik ini.

Dalam bab-bab selanjutnya akan dikenalkan berbagai macam metode sampling yang sesuai untuk permasalahan yang kita hadapi.

3.6. Diskusi: Probabilitas Seleksi

Pada proses seleksi *roulette wheel* di subbab sebelumnya dihitung nilai probabilitas (*prob*) dan probabilitas kumulatif (*probCum*) berdasarkan nilai *fitness*. Karena yang dihadapi adalah masalah maksimasi maka nilai *fitness* dihitung secara langsung *fitness= f(x)*. Rumusan ini bisa digunakan jika tidak ada nilai *f(x)* yang negatif. Jika ada nilai *f(x)* yang negatif maka rumus menghitung *fitness* bisa diubah sebagai berikut:

$$fitness = \frac{f(x) - f_{min}(x)}{f_{max}(x) - f_{min}(x)} \quad (3.1)$$

$f_{min}(x)$ dan $f_{max}(x)$ merupakan nilai terkecil dan terbesar dari fungsi obyektif pada generasi tersebut. Contoh:

	f(x)	fitness	prob	probCum
P_1	-1	0,000	0,000	0,000
P_2	0	0,200	0,100	0,100
P_3	3	0,800	0,400	0,500
P_4	4	1,000	0,500	1,000
Total		2,000		

$f_{min}(x)$=-1 dan $f_{max}(x)$=4

Perhatikan tabel di atas. Individu terbaik akan selalu mempunyai *fitness*=1 dan individu terburuk selalu mempunyai *fitness*=0. Sebagai akibatnya individu dengan nilai fungsi obyektif terkecil tidak akan pernah terpilih karena nilai probabilitasnya 0. Beberapa penelitian menunjukkan bahwa bisa jadi individu dengan nilai *fitness* lebih kecil justru menempati area yang lebih dekat dengan titik optimum. Hal ini biasanya terjadi pada optimasi fungsi yang mempunyai banyak titik optimum lokal. Berdasarkan kondisi ini maka rumus (3.1) bisa dimodifikasi sebagai berikut (Gen & Cheng 1997):

$$fitness = \frac{f(x) - f_{min}(x) + c}{f_{max}(x) - f_{min}(x) + c} \tag{3.2}$$

$0 < c < 1$

Perhatikan variasi nilai fitness untuk berbagai nilai c berikut:

	f(x)	fitness				
		c=0	*c*=0.1	*c*=0.3	*c*=0.5	*c*=0.7
P_1	-1	0,000	0,020	0,057	0,091	0,123
P_2	0	0,200	0,216	0,245	0,273	0,298
P_3	3	0,800	0,804	0,811	0,818	0,825
P_4	4	1,000	1,000	1,000	1,000	1,000

Semakin besar nilai *c* akan meningkatkan peluang individu terburuk untuk terpilih. Untuk masalah minimasi maka bisa digunakan rumus:

$$fitness = \frac{f_{min}(x) - f(x) + c}{f_{max}(x) - f_{min}(x) + c} \tag{3.3}$$

Rumus (3.3) menjamin individu dengan nilai fungsi obyektif lebih kecil akan mempunyai nilai *fitness* yang lebih besar.

3.7. Diskusi: Penanganan Konvergensi Dini

Perhatikan masalah konvergensi dini yang terjadi pada Sub-Bab 3.1.4. Hampir semua individu bernilai sama sebelum tercapainya titik optimum yang diinginkan. Ada banyak metode untuk mengatasi masalah ini. Satu metode sederhana yang bisa diterapkan adalah dengan melakukan *random injection* yaitu proses seleksi hanya memilih *popSize*-*n* individu (*n* =1 ..3). *n* individu terakhir dibangkitkan secara random seperti pada saat inisialisasi (Mahmudy, Marian & Luong 2013d, 2013e). *n*=0,1×*popSize* biasanya sudah cukup memadai. Untuk *popSize*=10, dengan memasukkan 1 individu random ini maka keragaman populasi akan tetap terjaga karena individu ini juga terlibat dalam proses reproduksi. Untuk menghemat waktu komputasi, pemasukan individu random ini tidak harus pada setiap generasi tapi bisa dilakukan setiap *g* interval generasi. Penentuan nilai *g* yang sesuai dilakukan melalui beberapa percobaan pendahuluan.

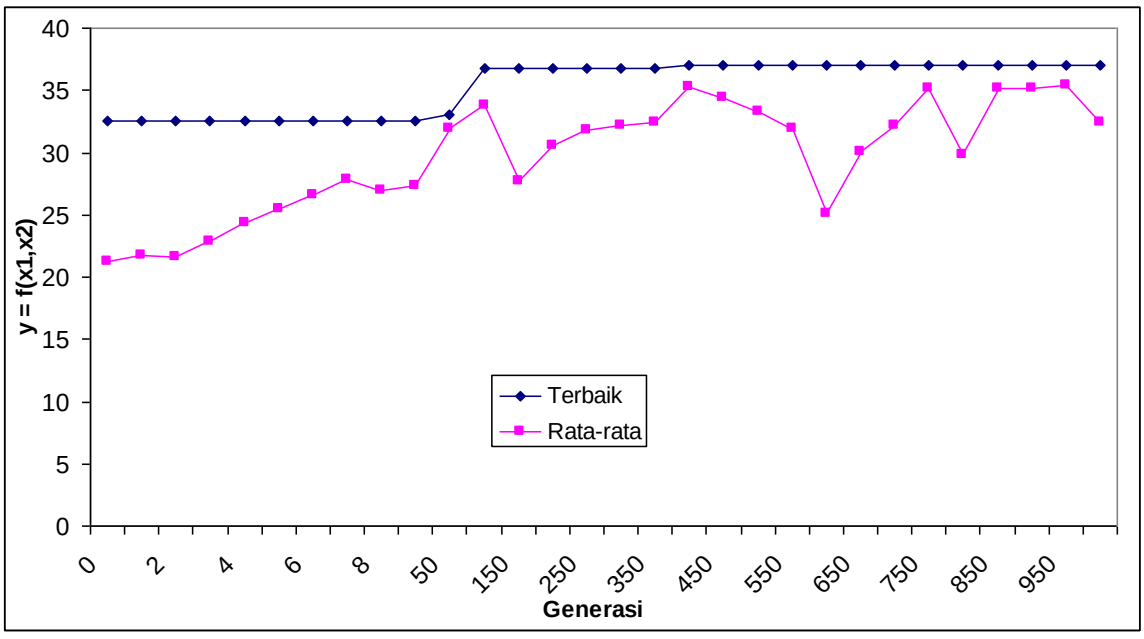
Dengan mengacu struktur GAs murni pada sub-bab sebelumnya maka teknik penanganan konvergensi dini dengan *random injection* bisa disusun sebagai berikut:

```
procedure AlgoritmaGenetika
begin
  t = 0
  inisialisasi P(t)
  while (bukan kondisi berhenti) do
    reproduksi C(t) dari P(t)
    evaluasi P(t) dan C(t)
    seleksi P(t+1) dari P(t) dan C(t)
    if (t mod g = 0)
      replace n individu
    end
    t = t + 1
  end while
end
```

Berikut ini adalah hasil penggunaan *random injection* 1 individu untuk kasus pada Sub-Bab 2.4.

generasi	$f(x_1,x_2)$	
	terbaik	rata-rata
0	32,5606	21,2507
1	32,5606	21,7481
2	32,5606	21,6480
3	32,5606	22,8637
4	32,5606	24,3919
5	32,5606	25,4174
6	32,5606	26,5430
7	32,5606	27,7997
8	32,5606	26,9720
9	32,5606	27,3259
50	33,0101	31,9865
100	36,7112	33,7484
150	36,7112	27,7559
200	36,7112	30,5534
250	36,7112	31,7417

generasi	$f(x_1,x_2)$	
	terbaik	rata-rata
300	36,7112	32,2169
350	36,7112	32,4219
400	37,0110	35,3276
450	37,0112	34,4334
500	37,0112	33,2982
550	37,0112	31,9309
600	37,0112	25,1292
650	37,0112	30,1114
700	37,0112	32,1155
750	37,0112	35,1785
800	37,0112	29,8486
850	37,0112	35,1296
900	37,0113	35,1574
950	37,0113	35,4518
1000	37,0113	32,4105



Gambar 3.5. Solusi RCGA pada tiap generasi menggunakan random injection satu individu

Metode ini (*random injection*) terbukti menghasilkan individu-individu yang lebih bervariasi dan juga nilai fungsi *fitness* yang lebih besar.

Cara lain yang bisa diterapkan adalah dengan melakukan pengujian konvergensi populasi yang diterapkan secara periodik sepanjang generasi. Jika nilai keragaman populasi dibawah nilai threshold maka mutasi akan dilakukan terhadap sebagaian chromosome. Gambar 3.6 menunjukkan perhitungan kemiripan dari dua kromosom dengan reprentasi biner atau bilangan bulat (*integer*). Setiap gen kromosom dibandingkan dan rasio kemiripan (*simmilarity ratio*) dinyatakan dalam persentase.

```
PROCEDURE Simmilarity
Input:  chromosome1, chromosome2
Output: simmilarity_ratio

length ← LengthOf (chromosomes1)
sim ← 0
FOR i=1 TO length DO
  IF chromosome1[i]= chromosome2[i] THEN
    sim ← sim + 1
  END IF
NEXT i
simmilarity_ratio ← sim/length * 100
RETURN simmilarity_ratio
END PROCEDURE
```

Gambar 3.6. Pseudo-code perhitungan rasio kemiripan dua kromosom

Gambar 3.7 menunjukkan perhitungan nilai keragaman (*diversity*) dari populasi (*pop*). Setiap kromosom dalam populasi dibandingkan dan dihitung rasio kemiripannya. Nilai keragaman didapatkan dengan mengurangi 1 dengan rata-rata rasio kemiripan.

```
PROCEDURE PopulationSimmilarity
Input: pop, pop_size
Ouput: diversity

total ← 0
n ← 0
FOR i=1 TO popSize-1 DO
  FOR j=i+1 TO popSize DO
    total ← total + Simmilarity (pop[i], pop[j])
    n ← n + 1
  NEXT j
NEXT i
diversity ← 1-total/n
END PROCEDURE
```

Gambar 3.7. Pseudo-code perhitungan nilai keragaman populasi

Beberapa metode yang lebih kompleks untuk penanganan konvergensi dini akan dibahas dalam bab berikutnya.

3.8. Rangkuman

Pada bab ini telah dibahas struktur GAs menggunakan pengkodean chromosome bilangan pecahan (*real-coded chromosome*). Metode reproduksi yang dibahas adalah *extended intermediate crossover* dan *random mutation*. Dibahas juga metode *one-cut-point crossover* yang digunakan pada pengkodean chromosome biner bisa dengan mudah diterapkan pada pengkodean real.

Metode seleksi yang dibahas adalah *binary tournament selection*. Metode seleksi lain yang sering digunakan dalam penelitian, *elitism selection* dan *replacement selection*, juga dibahas beserta contoh penerapannya. Untuk semua metode seleksi yang dibahas diberikan juga *pseudo-code*-nya.

Bab ini ditutup dengan diskusi penentuan parameter GAs yang tepat, mekanisme sampling yang digunakan pada proses seleksi, penyesuaian (*adjustment*) probabilitas seleksi, dan penggunaan *random injection* untuk menangani konvergensi dini.

3.9. Latihan

Untuk memperjelas pemahaman anda, kerjakanlah latihan berikut sebisa mungkin tanpa melihat materi pada buku!

1. Sebutkan kelemahan algoritma genetika dengan pengkodean biner jika digunakan pada optimasi fungsi!
2. Misalkan yang terpilih sebagai induk adalah P_1=(2,3, 5,2) dan P_2=(4,8, 3,1). Jika a=[0,1, 0,2] tentukan dua *offspring* (C_1 dan C_2) yang terbentuk dari *extended intermediate crossover*!
3. Misalkan yang terpilih sebagai induk adalah P=(2,3, 5,2) dan nilai x_2 berada pada range [1,0, 10,0]. Jika r=0,01 dan gen yang terpilih nomer 2, tentukan *offspring C* yang terbentuk dari *random mutation*!
4. Misalkan terdapat himpunan individu dalam populasi dengan *popSize*=5 sebagai berikut:

individu	fitness
P_1	9
P_2	10
P_3	2
P_4	9
P_5	6

Terdapat juga himpunan *offspring* sebagai berikut:

individu	fitness
C_1	4
C_2	8
C_3	7

Tentukan himpunan individu yang lolos ke generasi selanjutnya jika digunakan *elitism selection*!

5. Misalkan terdapat himpunan individu dalam populasi dengan *popSize*=5 sebagai berikut:

individu	fitness
P_1	9
P_2	10
P_3	2
P_4	9
P_5	6

Terdapat juga himpunan *offspring* sebagai berikut:

individu	parent	fitness
C_1	P_1 dan P_3	4
C_2	P_3 dan P_5	8
C_3	P_4	5

Tentukan himpunan individu yang lolos ke generasi selanjutnya jika digunakan *replacement selection*!

6. Sebutkan keuntungan dan kerugian jika nilai parameter ukuran populasi (*popSize*), probabilitas *crossover* (*pc*) dan probabilitas mutasi (*pm*) ditentukan semakin besar?
7. Pada proses seleksi terdapat mekanisme sampling untuk memilih individu yang dipertahankan hidup. Sebutkan tiga kategori metode dasar untuk melakukan sampling!
8. Apa tujuan dari penyesuaian (*adjustment*) probabilitas seleksi?

BAB 4

Optimasi Masalah Kombinatorial

4.1. Pengantar

Masalah kombinatorial adalah masalah yang mempunyai himpunan solusi *feasible* yang terhingga. Meskipun secara prinsip solusi dari masalah ini bisa didapatkan dengan enumerasi lengkap, pada masalah kompleks dibutuhkan waktu yang tidak bisa diterima secara praktis (Gen & Cheng 2000). Sebagai contoh pada masalah *Travelling Salesperson Problem* (*TSP*) yang melibatkan pemilihan rute terbaik untuk mengunjungi *n* kota. Metode enumerasi lengkap harus menguji *n!* kemungkinan solusi. Untuk masalah sederhana dengan n=20 ada lebih dari $2{,}4\times10^{18}$ kemungkinan solusi. Sebuah *personal computer* mungkin memerlukan waktu lebih dari 5 jam untuk melakukan enumerasi lengkap (Mahmudy 2006), sebuah hal yang tidak bisa diterima secara praktis.

Algoritma genetika telah sukses diterapkan pada berbagai masalah kombinatorial seperti perencanaan dan penjadwalan produksi pada industry manufaktur (Mahmudy, Marian & Luong 2012b, 2013b, 2013e). Meskipun solusi optimum tidak diperoleh, tetapi solusi yang mendekati optimum bisa didapatkan dalam waktu yang relatif cepat dan bisa diterima secara praktis. Bab ini membahas berbagai permasalahan kombinatorial sederhana yang banyak ditemui di kehidupan nyata.

4.2. Travelling Salesman Problem (TSP)

Perncarian rute terbaik merupakan salah satu permasalahan yang sering dihadapi dalam kehidupan sehari-hari. Salah satu contoh yaitu rute manakah yang memiliki biaya paling murah (atau paling pendek) untuk dilalui seorang *salesman* yang harus mengunjungi sejumlah daerah, tiap daerah harus dikunjungi tepat satu kali kemudian kembali lagi ke tempat semula. Permasalahan tersebut dikenal sebagai *Travelling Salesman Problem*

(TSP). Jika terdapat lebih dari seorang *salesman* maka disebut *multi Travelling Salesman Problem (m-TSP)*.

Secara matematis *TSP* bisa diformulasikan sebagai masalah minimasi biaya perjalanan sebagai berikut:

$$Z = \min \left\{\sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} x_{ij}\right\} \quad (4.1)$$

dengan kendala (*contraint*):

$$\sum_{i=1}^{n} x_{ij} = 1, \quad \text{untuk } j = 1,2,3, \ldots, n-1 \quad (4.2)$$

$$\sum_{j=1}^{n} x_{ij} = 1, \quad \text{untuk } i = 1,2,3, \ldots, n-1 \quad (4.3)$$

n menyatakan banyaknya kota (selanjutnya disebut simpul/*node*).

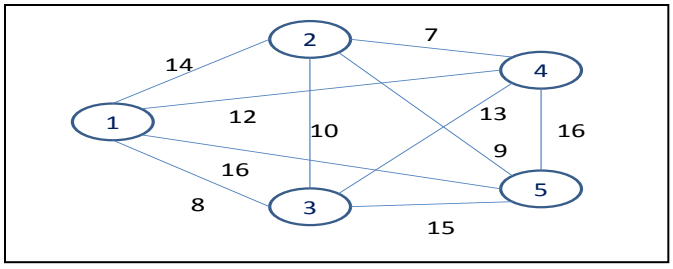
$x_{ij} = 1$ apabila ada perjalanan *salesman* dari simpul *i* menuju simpul *j*, 0 jika tidak ada perjalanan.

c_{ij} menyatakan biaya (atau jarak, tergantung tujuan minimasi) dari simpul *i* menuju simpul *j*.

Persamaan (4.2) dan Persamaan (4.3) menjamin bahwa setiap simpul hanya dikunjungi sekali oleh *salesman*.

4.2.1. Representasi Chromosome

Perhatikan masalah TSP pada Gambar 4.1. Ada 4 simpul (kota) yang terhubung dan angka pada busur menunjukkan jarak antar simpul.



Gambar 4.1. Contoh masalah TSP

Dari Gambar 4.1 bisa dibuat tabel jarak antar simpul sebagai berikut:

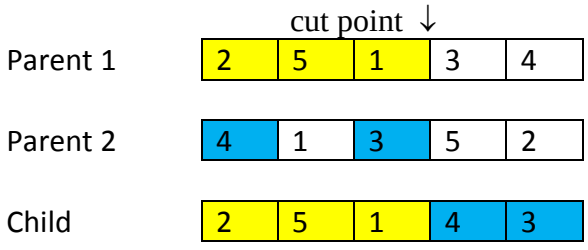
Node	1	2	3	4	5
1	-	14	8	12	16
2	14	-	10	7	9
3	8	10	-	13	15
4	12	7	13	-	16
5	16	9	15	16	-

Representasi permutasi bisa digunakan untuk menyatakan sebuah solusi. Setiap gen pada chromosome berupa angka integer yang menyatakan nomer dari tiap simpul. Sebuah chromosome [2 3 4 1 5] menyatakan bahwa perjalanan dimulai dari simpul 2 kemudian secara berurutan mengunjungi simpul 3, 4, 1, 5 dan kemudian kembali ke simpul 2. Berikut ini contoh beberapa chromosome dan total jarak antar simpul beserta nilai fitnessnya:

No	Chromosome	Total Jarak (*J*)	$fitness = \frac{100}{J}$
1	[1 2 3 4 5]	14+10+13+16+16=69	1,449
2	[2 3 4 1 5]	10+13+12+16+9=60	1,667
3	[4 1 2 5 3]	12+14+9+15+13=63	1,587

4.2.2. Crossover

Metode crossover paling sederhana adalah dengan melakukan modifikasi *one-cut-point crossover* yang digunakan pada representasi biner. Perhatikan contoh pada Gambar 4.2. Segment kiri dari chromosome *child* didapatkan dari *parent 1* dan segmen kanan didapatkan dari urutan gen tersisa dari *parent 2*.

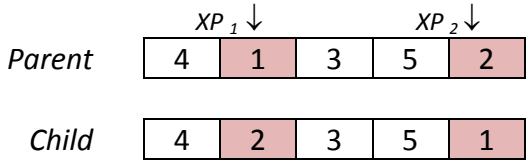


Gambar 4.2. Crossover pada representasi permutasi

Beberapa metode lain yang digunakan pada representasi permutasi adalah *partial-mapped crossover* (PMX), *order crossover* (OX), *cycle crossover* (CX), *position-based crossover*, *order-based crossover*, dan *heuristic crossover* (Gen & Cheng 1997).

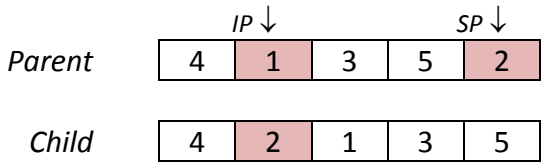
4.2.3. Mutasi

Metode mutasi yang paling sederhana adalah *reciprocal exchange mutation*. Metode ini bekerja dengan memilih dua posisi (*exchange point* / XP) secara random kemudian menukarkan nilai pada posisi tersebut seperti pada Gambar 4.3.



Gambar 4.3. Reciprocal exchange mutation

Metode lain yang bisa digunakan adalah *insertion mutation*. Metode ini bekerja dengan memilih satu posisi (*selected point* / *SP*) secara random kemudian mengambil dan menyisipkan nilainya pada posisi lain (*insertion point* / *IP*) secara random seperti pada Gambar 4.4.



Gambar 4.4. Insertion mutation

4.3. Flow-Shop Scheduling Problem (FSP)

FSP berkaitan dengan penjadwalan sejumlah *j* job pada sejumlah *m* mesin. Semua job mempunyai urutan pemrosesan (operasi) yang sama, mulai dari mesin ke-1, mesin ke-2, dan seterusnya sampai mesin ke-*m*. Waktu pemrosesan setiap operasi pada sebuah mesin mungkin berbeda dan diasumsikan telah diketahui sebelumnya.

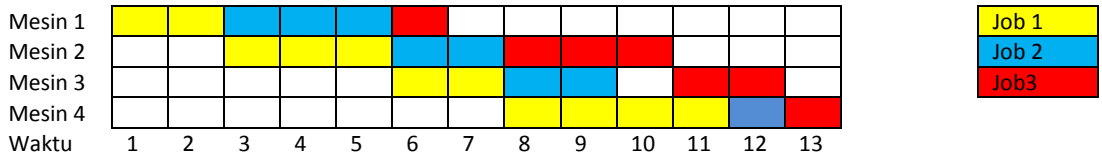
Kendala lengkap dari FSP bisa diringkas sebagai berikut:

- Sebuah job hanya mengunjungi sebuah mesin tepat satu kali.
- Tidak ada kendala *precedence* di antara operasi-operasi dari job yang berbeda.
- Operasi job pada mesin tidak bisa dinterupsi.
- Sebuah mesin hanya bisa memproses satu operasi pada satu waktu.

Urutan job yang harus diselesaikan menentukan waktu selesainya seluruh job (*makespan*). Proses optimasi dilakukan untuk menentukan urutan operasi yang menghasilkan nilai *makespan* minimum. Perhatikan masalah penjadwalan 3 job (J1, J2, dan J3) pada 4 mesin yang mempunyai waktu pemrosesan sebagai berikut:

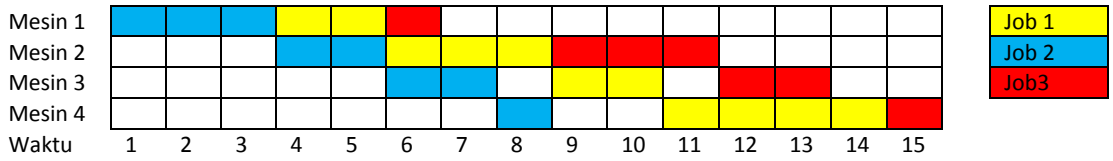
Job	Mesin			
	1	**2**	**3**	**4**
1	2	3	2	4
2	3	2	2	1
3	1	3	2	1

Jika pemrosesan job dilakukan dengan urutan J1 → J2 → J3 maka didapatkan makespan sebesar 13 yang ditunjukan dalam Gantt-Chart pada Gambar 4.5.



Gambar 4.5. Gantt-Chart untuk urutan job J1 → J2 → J3

Jika pemrosesan job dilakukan dengan urutan J2 → J1 → J3 maka didapatkan makespan sebesar 15 yang ditunjukan dalam Gantt-Chart pada Gambar 4.6.



Gambar 4.6. Gantt-Chart untuk urutan job J2 → J1 → J3

Representasi permutasi seperti yang digunakan pada TSP bisa diadopsi untuk masalah FSP. Setiap gen pada chromosome menyatakan nomer dari tiap job. Operator crossover dan mutasi yang sama pada TSP juga bisa digunakan.

4.4. Two-Stage Assembly Flow-Shop Scheduling Problem

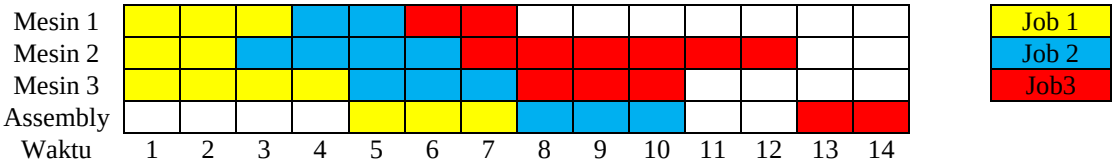
Two-stage assembly flowshop merupakan variasi dari FSP. Pada permasalahan ini sebuah job memiliki *m* operasi yang bisa dikerjakan pada *m* mesin secara paralel. Pemrosesan tahap kedua (*assembly stage*) hanya bisa dilakukan setelah semua operasi pada tahap pertama telah diselesaikan (Allahverdi & Al-Anzi 2008).

Ilustrasi sederhana dari masalah ini adalah pada proses pembuatan sebuah *personal computer* (PC) yang bisa dianggap sebagai sebuah job. Setelah sejumlah komponen seperti CPU, harddisk, memory dan lain-lain selesai dibuat pada tahap pertama pada tempat/mesin yang berbeda secara paralel maka komponen-komponen ini masuk ke *assembly-station* pada tahap kedua untuk dirakit sesuai spesifikasi yang dibutuhkan konsumen. Jika pada saat yang sama terdapat *n* pesanan PC dengan spesifikasi yang berbeda maka bisa dikatakan ada *n* job. Masalah yang timbul adalah bagaimana menentukan urutan pembuatan semua pesanan PC supaya didapatkan waktu penyelesaian semua PC dalam waktu yang paling singkat

Misalkan terdapat 3 job yang harus diproses pada 3 mesin dengan waktu operasi sebagai berikut:

job	mesin			assembly
	1	2	3	
1	3	2	4	3
2	2	4	3	3
3	2	6	3	2

Misalkan urutan pemrosesan job ditentukan job 1 $\rightarrow$ job 2 $\rightarrow$ job 3. Pada job 1, operasi tahap kedua (*assembly*) bisa dilakukan pada waktu ke-4 setelah setelah semua operasi tahap pertama diselesaikan. Gantt-chart dari urutan operasi tersebut ditunjukan pada Gambar 4.7 dengan nilai makespan=14.



Gambar 4.7. Gantt-chart untuk two-stage assembly flowshop

Seperti halnya pada FSP, representasi permutasi seperti yang digunakan pada TSP bisa diadopsi untuk masalah *two-stage assembly flowshop*. Operator crossover dan mutasi yang sama pada TSP juga bisa digunakan.

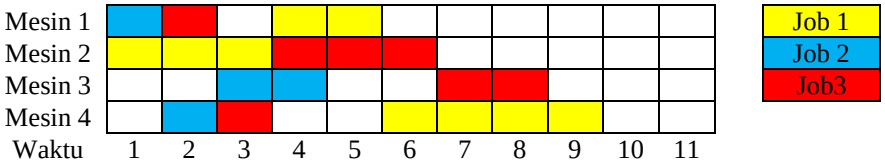
4.5. Job-Shop Scheduling Problem (JSP)

JSP merupakan perluasan (bentuk umum) dari FSP. Pada masalah ini tiap job mungkin mempunyai urutan operasi yang berbeda. Pada beberapa kasus, sebagian job hanya memerlukan sebagian mesin. Perhatikan masalah JSP berikut:

Job	Waktu Operasi Pada Mesin				Urutan Operasi
	1	2	3	4	
1	2	3	-	4	$2 \rightarrow 1 \rightarrow 4$
2	1	-	2	1	$1 \rightarrow 4 \rightarrow 3$
3	1	3	2	1	$1 \rightarrow 4 \rightarrow 2 \rightarrow 3$

Pada kasus ini, job 1 tidak memerlukan mesin 3 dan urutan operasinya dimulai dari mesin 2, kemudian ke mesin 1 dan 4.

Misalkan $O_{i,j}$ menyatakan operasi ke-j dari job i. Jika pemrosesan job dilakukan dengan urutan $O_{1,1} \rightarrow O_{2,1} \rightarrow O_{1,2} \rightarrow O_{3,1} \rightarrow O_{3,2} \rightarrow O_{1,3} \rightarrow O_{2,2} \rightarrow O_{3,3} \rightarrow O_{3,4} \rightarrow O_{2,3}$ maka didapatkan makespan sebesar 9 yang ditunjukan dalam Gantt-Chart pada Gambar 4.8.



Gambar 4.8. Gantt-chart untuk JSP

4.5.1. Representasi Chromosome

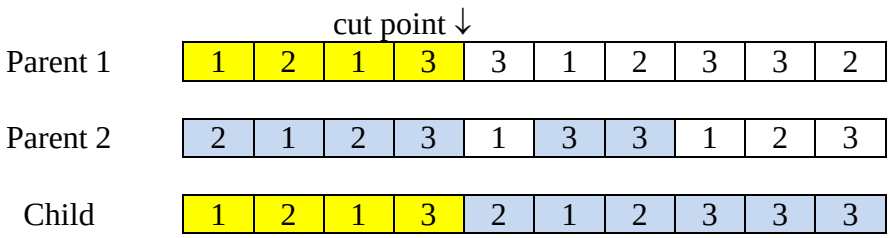
Representasi integer yang memuat nomer job (*job-based representation*) bisa digunakan untuk masalah JSP. Misalkan untuk solusi di atas maka bisa dinyatakan sebagai:

[1 2 1 3 3 1 2 3 3 2]

Perhatikan pada representasi ini angka 1 muncul sebanyak 3 kali yang menyatakan bahwa job 1 mempunyai 3 operasi. Hal serupa terjadi pada angka 3 yang muncul sebanyak 4 kali yang menyatakan bahwa job 3 mempunyai 4 operasi.

4.5.2. Crossover

Modifikasi *one-cut-point crossover* bisa diterapkan pada *job-based representation*. Perhatikan contoh pada Gambar 4.9. Segment kiri dari chromosome *child* didapatkan dari *parent 1* dan segmen kanan didapatkan dari urutan gen tersisa dari *parent 2*.



Gambar 4.9. Crossover pada *job-based representation*

4.5.3. Mutasi

Metode *reciprocal exchange mutation* dan *insertion mutation* yang digunakan pada representasi permutasi bisa dengan mudah diterapkan untuk *job-based representation*.

4.5.4. Representasi Permutasi Untuk JSP

Dengan penanganan khusus, representasi permutasi bisa diterapkan untuk JSP. Untuk permasalahan JSP yang telah diuraikan sebelumnya, diperlukan peta sebagai berikut:

Angka permutasi	1	2	3	4	5	6	7	8	9	10
Job	1	1	1	2	2	2	3	3	3	3

Gambar 4.10. Representasi permutasi untuk JSP

Peta pada Gambar 4.10 menunjukkan aturan konversi dari angka permutasi yang muncul pada chromosome ke indeks dari tiap job. Misalkan didapatkan chromosome dengan representasi permutasi sebagai berikut:

[7 1 8 4 10 2 6 9 3 5]

Maka bisa dikonversi menjadi *job-based representation* sebagai berikut:

[3 1 3 2 3 1 2 3 1 2]

Dengan representasi permutasi ini, operator crossover dan mutasi yang sama pada TSP juga bisa digunakan.

4.6. Transportation Problem

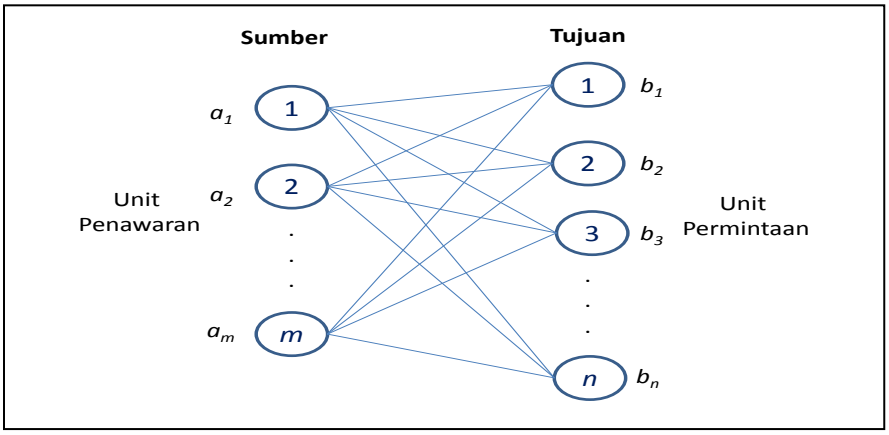
Persoalan transportasi berkaitan dengan pendistribusian suatu komoditas atau produk dari sejumlah sumber (*supply*) kepada sejumlah tujuan (*destination*, *demand*) dengan

tujuan meminimumkan ongkos pengangkutan yang terjadi. Persoalan ini mempunyai beberapa karakteristik yang bisa diringkas sebagai berikut (Taha 2011):

1. Terdapat sejumlah sumber dan sejumlah tujuan tertentu.
2. Kuantitas komoditas yang didistribusikan dari setiap sumber dan yang diminta oleh setiap tujuan besarnya tertentu.
3. Komoditas yang dikirim atau diangkut dari suatu sumber ke suatu tujuan besarnya sesuai dengan permintaan dan atau kapasitas sumber.
4. Ongkos pengangkutan komoditas dari suatu sumber ke suatu tujuan besarnya tertentu.

Persoalan transportasi merupakan kasus khusus dari model pemrograman linier (*linear programming*). Persoalan ini membutuhkan pembatas (*constrains*) dan variabel yang sangat banyak sehingga penggunaan komputer dalam penyelesaiannya dengan model matematis (misalnya dengan metode simplex) memerlukan perhitungan yang panjang dan tidak praktis. GAs terbukti efektif untuk mendapatkan solusi yang mendekati optimum dalam waktu yang relatif cepat (Mahmudy 2007).

Model transportasi sederhana dari sebuah jaringan dengan *m* sumber dan *n* tujuan digambarkan sebagai berikut (Taha 2011):



Gambar 4.11. Model Transportasi

Gambar 4.11 menyatakan bahwa sumber *i* (*i* = 1, 2, …, *m*) mempunyai persediaan a_i unit untuk didistribusikan ke tujuan-tujuan, dan tujuan *j* (*j* = 1, 2, …, *n*) mempunyai permintaan b_j unit untuk diterima dari sumber-sumber.

Variabel c_{ij} (*i* = 1, 2, …, *m*; *j* = 1, 2, …, *n*) adalah biaya pendistribusian yang dibutuhkan dari sumber *i* ke tujuan *j*. x_{ij} (*i* = 1, 2, …, *m*; *j* = 1, 2, …, *n*) adalah variabel keputusan yang menyatakan banyaknya produk yang harus dikirimkan dari sumber *i* ke tujuan *j*.

Dari gambaran yang telah diberikan, permasalahan transportasi dapat dinyatakan secara matematis sebagai berikut:

$$Z = \min \left\{\sum_{i=1}^{m}\sum_{j=1}^{n} c_{ij}x_{ij}\right\} \quad (4.4)$$

dengan kendala (*contraint*):

$$\sum_{j=1}^{n} x_{ij} = a_i, \quad \text{untuk } i = 1,2,3,\dots,m \quad (4.4)$$

$$\sum_{i=1}^{m} x_{ij} = b_j, \quad \text{untuk } j = 1,2,3,\dots,n \quad (4.5)$$

$$x_{ij} \geq 0, \quad \text{untuk semua } i \text{ dan } j \quad (4.5)$$

Persamaan (4.4) menyatakan bahwa jumlah barang yang dikirim dari setiap sumber sama dengan ketersediaan barang pada sumber tersebut. Persamaan (4.5) menyatakan bahwa jumlah barang yang dikirim ke setiap tujuan sama dengan permintaan barang pada tujuan tersebut. Kendala-kendala yang diberikan menyatakan bahwa penawaran total sama dengan permintaan total yang biasa disebut model transportasi seimbang.

4.6.1. Representasi Chromosome

Misalkan diberikan permasalahan transportasi dengan 3 sumber dan 4 tujuan. Persediaan di sumber adalah 10, 15 dan 5. Permintaan di tujuan adalah 10, 5, 5 dan 10.

Matriks biaya adalah sebagai berikut:

$$C = \begin{bmatrix} 4 & 5 & 1 & 2 \\ 3 & 2 & 4 & 5 \\ 5 & 4 & 3 & 1 \end{bmatrix}$$

Karena solusi permasalahan transportasi adalah matriks yang menyatakan banyaknya produk yang harus dikirimkan dari sumber *i* ke tujuan *j* maka sebuah chromosome dapat dinyatakan sebagai matriks dengan contoh berikut:

					a_i
	0	0	5	5	**10**
P =	10	5	0	0	**15**
	0	0	0	5	**5**
b_j	**10**	**5**	**5**	**10**	

Perhatikan total elemen setiap baris sama dengan banyaknya persediaan di tiap sumber dan total elemen setiap kolom sama dengan banyaknya pemintaan di tiap tujuan. Total biaya dari solusi di atas adalah 5×1+5×2+10×3+5×2+5×1=60.

4.6.2. Crossover

Crossover dilakukan dengan melakukan perhitungan rata-rata tiap elemen dari dua chromosome (P_1 dan P_2) untuk menghasilkan anak (C) seperti contoh berikut:

$$P_1 = \begin{bmatrix} 0 & 0 & 5 & 5 \\ 10 & 5 & 0 & 0 \\ 0 & 0 & 0 & 5 \end{bmatrix}$$

$$P_2 = \begin{bmatrix} 5 & 0 & 0 & 5 \\ 0 & 5 & 5 & 5 \\ 5 & 0 & 0 & 0 \end{bmatrix}$$

					a_i
	3	0	3	5	**11**
C =	5	5	3	3	**16**
	3	0	0	3	**6**
b_j	**11**	**5**	**6**	**11**	

Karena ada proses pembulatan ke atas maka dihasilkan chromosome yang infeasible. Perhatikan total nilai tiap baris dan kolom yang tidak sama dengan persediaan dan permintaan. Mekanisme perbaikan (*repairing*) diperlukan untuk menghasilkan chromosome yang feasible sebagai berikut:

					a_i
	2	0	3	5	**10**
C' =	5	5	2	3	**15**
	3	0	0	2	**5**
b_j	**10**	**5**	**5**	**10**	

Perbaikan chromosome di atas dilakukan dengan melakukan perubahan pada tiap sel yang total nilai baris dan kolomnya tidak sesuai.

4.6.3. Mutasi

Metode mutasi sederhana bekerja dengan memilih 4 titik secara acak sehingga membentuk loop tertutup. Alokasi barang pada tiap titik sudut loop diubah sedemikian rupa sehingga total tiap baris dan total tiap kolom tidak berubah (Mahmudy 2007).

					a_i
	0	0	5	5	**10**
P =	10	5	0	0	**15**
	0	0	0	5	**5**
b_j	**10**	**5**	**5**	**10**	

					a_i
	0	0	5	5	**10**
C =	5	5	0	5	**15**
	5	0	0	0	**5**
b_j	**10**	**5**	**5**	**10**	

4.6.4. Representasi Permutasi

Representasi permutasi bisa diadopsi untuk permasalahan transportasi. Misalkan untuk contoh kasus pada Subbab 4.6.1, setiap sel pada matriks alokasi diberi nomer urut seperti contoh berikut:

1	2	3	4
5	6	7	8
9	10	11	12

Setiap gen pada chromosome menyatakan prioritas alokasi. Untuk contoh chromosome:

[5 8 3 1 9 12 10 11 6 2 7 4]

maka bisa diuraikan menjadi sebuah solusi dengan langkah-langkah berikut:

1. Alokasikan unit maksimum pada sel dengan nomor urut 5. Pada sel ini unit maksimum yang bisa dialokasikan sebesar 10. a'_i dan b'_j menunjukkan total baris dan kolom sementara.

					a'_i	a_i
	1	2	3	4		**10**
	5	6	7	8	**10**	**15**
	9	10	11	12		**5**
b'_j	**10**					
b_j	**10**	**5**	**5**	**10**		

2. Isi sel 8 dengan unit maksimum sebesar 5.

					a'_i	a_i
	1	2	3	4		**10**
	5	6	7	8	**15**	**15**
	9	10	11	12		**5**
b'_j	**10**			**5**		
b_j	**10**	**5**	**5**	**10**		

3. Isi sel 3 dengan unit maksimum sebesar 5.

					a'_i	a_i
	1	2	3	4		
			5		**5**	**10**
	5	6	7	8		
	10			5	**15**	**15**
	9	10	11	12		
						5
b'_j	**10**		**5**	**5**		
b_j	**10**	**5**	**5**	**10**		

4. Sesuai urutan gen pada chromosome, isi sel 1. Tetapi sel ini tidak bisa disi karena kolom 1 sudah penuh ($b'_1 = b_1$). Hal yang sama juga terjadi pada sel 9 karena kolom yang bersesuaian sudah penuh.

5. Isi sel 12 dengan unit maksimum sebesar 5.

					a'_i	a_i
1	2	3	4			
		5			**5**	**10**
5	6	7	8			
10			5		**15**	**15**
9	10	11	12			
			5		**5**	**5**
b'_j	**10**		**5**	**10**		
b_j	**10**	**5**	**5**	**10**		

6. Sel 10 dan 11 tidak bisa disi karena baris 3 sudah penuh ($a'_3 = a_3$). Hal yang sama juga terjadi pada sel 6 karena baris 2 sudah penuh.
7. Isi sel 2 dengan unit maksimum sebesar 5. Setelah sel ini disi maka alokasi sudah komplit ($a'_i=a_i$ dan $b'_j=b_j$) sehingga pengecekan gen setelahnya tidak perlu dilakukan.

					a'_i	a_i
1	2	3	4			
	5	5			**10**	**10**
5	6	7	8			
10			5		**15**	**15**
9	10	11	12			
			5		**5**	**5**
b'_j	**10**	**5**	**5**	**10**		
b_j	**10**	**5**	**5**	**10**		

4.7. Flexible Job-Shop Scheduling Problem (FJSP)

Flexible job-shop problem (FJSP) merupakan bentuk umum (*generalized form*) dari JSP klasik. Pada permasalahan FSP, sebuah job memiliki beberapa operasi. Sebuah operasi bisa dikerjakan pada beberapa pilihan mesin. Skenario ini lebih dekat dengan kasus nyata yang ditemui pada industri manufaktur (Zhang et al. 2009). Keberadaan mesin alternatif ini membuat FJSP lebih sulit diselesaikan dibandingkan JSP.

Ada dua keputusan yang harus dibuat pada FJPS. Keputusan yang pertama adalah penentuan mesin untuk tiap operasi (*routing problem*). Keputusan yang kedua adalah menentukan urutan operasi-operasi tersebut (*scheduling problem*). Seperti halnya

dengan JSP, tujuan utama dari FJSP adalah meminimumkan waktu selesainya seluruh job (*makespan*).

Karena ada dua keputusan yang harus dibuat, maka pendekatan untuk menyelesaikan FJPS bisa diklasifikasikan dalam dua kategori dasar, yaitu pendekatan hirarki (*hierarchical approaches*) dan pendekatan terintegrasi (*integrated approaches*).

Pendekatan hirarki digunakan untuk mengurangi kompleksitas permasalahan dengan menguraikan FJSP ke dalam dua sub permasalahan. Sub permasalahan yang pertama adalah penentuan mesin menggunakan metode heuristik atau *dispatching rules*. Sub permasalahan yang kedua adalah menentukan urutan operasi-operasi tersebut yang menghasilkan nilai *makespan* minimum. Pendekatan terintegrasi menyelesaikan kedua sub permasalahan tersebut secara bersamaan/simultan (Al-Hinai & ElMekkawy 2011; Pezzella, Morganti & Ciaschetti 2008; Yazdani, Amiri & Zandieh 2010). Meskipun pendekatan terintegrasi lebih sulit dan membutuhkan waktu komputasi yang lebih tinggi, sejumlah penelitian membuktikan bahwa pendekatan ini mampu memberikan hasil yang lebih baik.

4.7.1. Representasi Chromosome *Task Sequencing List*

Satu kasus FJSP diadopsi dari (Mahmudy, Marian & Luong 2013a) disajikan sebagai berikut:

job	operasi	machine	time
1	1	1	5
		2	6
	2	3	4
2	1	2	7
		3	8
	2	1	6
		3	4
	3	2	3
3	1	1	4
	2	2	4
4	1	1	5
	2	3	4
	3	1	6

Tabel di atas menunjukkan terdapat 4 job. Operasi 1 dari job 1 bisa dikerjakan di dua pilihan mesin. Jika dikerjakan di mesin 1 maka memerlukan waktu sebesar 5 unit. Jika dikerjakan di mesin 2 maka memerlukan waktu sebesar 6 unit.

Satu representasi yang dinamakan *task sequencing list representation* diusulkan oleh Kacem et al. (2002). Representasi ini memuat tiga bilangan bulat, yaitu: job, operasi dan mesin. Contoh dari chromosome untuk representasi ini ditunjukkan dalam Gambar 4.12.

job	2	4	1	2	3	1	3	4	2	4
operasi	1	1	1	2	1	2	2	2	3	3
mesin	2	1	2	1	1	3	2	3	2	1
urutan	1	2	3	4	5	6	7	8	9	10

Gambar 4.12. Contoh chromosome dari *task sequencing list representation*

Contoh chromosome dalam Gambar 4.12 menunjukan bahwa prioritas penjadwalan yang pertama adalah memproses operasi 1 dari job 2 pada mesin 2, diikuti dengan memproses operasi 1 dari job 4 pada mesin 1, dan seterusnya.

Representasi *task sequencing list* mensyaratkan bahwa operasi reproduksi menggunakan crossover dan mutasi harus dilaksanakan secara hati-hati untuk menghasilkan *offspring* yang feasible.

4.7.2. Representasi Chromosome Bilangan Pecahan

Mahmudy, Marian and Luong (2013a) mengajukan penggunaan vector bilangan real sebagai representasi chromosome untuk FJSP. Keunggulan representasi ini adalah berbagai jenis metode crossover dan mutasi bisa diterapkan. Representasi ini juga selalu menghasilkan solusi feasible. Properti ini sangat berguna untuk menghemat waktu komputasi yang dibutuhkan untuk proses perbaikan (*repairing*) chromosome infeasible.

Untuk proses decoding chromosome untuk kasus pada Sub-Bab 4.7.1, dibutuhan tabel yang memetakan indeks urutan kemunculan gen dengan nomer job sebagai berikut:

index	1	2	3	4	5	6	7	8	9	10
job	1	1	2	2	2	3	3	4	4	4

Berdasarkan tabel di atas maka sebuah chromosome P=[143, 209, 115,173, 75, 179, 193, 96 83, 144] bisa diuraikan menjadi sebuah solusi seperti ditunjukkan pada table berikut:

x	operasi	sorted *x*	operasi'	job,op,mac
143	1	75	5	2, 1, 3
209	2	83	9	4, 1, 1
115	3	96	8	4, 2, 3
173	4	115	3	2, 2, 3
75	5	143	1	1, 1, 2
179	6	144	10	4, 3, 1
193	7	173	4	2, 3, 2
96	8	179	6	3, 1, 1
83	9	193	7	3, 2, 2
144	10	209	2	1, 2, 3

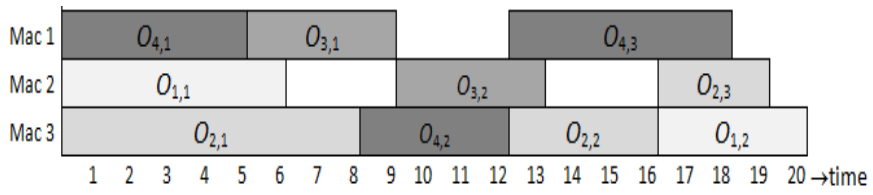
Langkah pertama untuk mengkonversi chromosome manjadi solusi adalah dengan mengurutkan ***x*** (bersama-sama ***operasi***) secara *ascending* sehingga dihasilkan ***sorted x*** dan ***operasi'***. Dari ***operasi'*** bisa didapatkan nomer ***job*** pada kolom terakhir dengan menggunakan table peta sebelumnya. Misal operasi' 5 menunjukkan job 2, operasi' 9 menunjukkan job 4, dan seterusnya. Operasi (***op***) pada kolom terakhir bisa didapatkan dengan memberi nomor urut kemunculan ***job*** yang sama.

Untuk menentukan mesin yang dipakai (***mac***) maka ***x*** dikonversi ke bilangan biner. Misalkan x_1 dikonversi ke $(1001011)_2$. 2 bit paling kanan $(11)_2$=3 dipakai untuk menentukan indeks mesin. Karena ada 2 alternatif mesin (*n*=2) yang bisa dipakai untuk operasi pertama dari job 2, maka digunakan rumus sebagai berikut:

$$indeks\ mesin = 3 \text{ MOD } n + 1 = 3 \text{ MOD } 2 + 1 = 2$$

MOD merupakan operator untuk menghitung sisa pembagian. Dari hasil ini maka operasi pertama dari job 2 dilakukan pada alternatif mesin yang ke-2, yaitu mesin 3. Pada proses kenversi ini digunakan 2 bit paling kanan karena maksimum banyaknya alternative mesin untuk setiap operasi adalah sebesar 2 yang hanya memerlukan 2 bit pada representasi biner. Untuk kasus lain dengan banyak alternatif mesin sebanyak 10 maka diperlukan 4 bit.

Dari solusi ini bisa dihasilkan Gantt-chart untuk menghitung makespan sebagai berikut:



Gambar 4.13. Gant-chart untuk menghitung makespan

4.8. Multi Travelling Salesman Problem (m-TSP)

Multi Travelling Salesman Problem (m-TSP) merupakan pengembangan dari TSP. Pada permasalahan ini terdapat lebih dari seorang *salesman.* Representasi permutasi yang digunakan untuk TSP bisa dimodifikasi sehingga bisa digunakan untuk mTSP. Modifikasi bisa dilakukan dengan menambahkan segmen untuk menunjukan banyaknya daerah yang dikunjungi oleh tiap salesman.

Misalkan terdapat 10 daerah yang harus dikunjungi (m=10) dan 3 orang salesman (n=3), maka sebuah chromosome bisa ditunjukkan seperti pada Gambar 4.14.

posisi	1	2	3	4	5	6	7	8	9	10	11	12	13
gen	2	5	1	7	6	10	4	9	8	3	3	4	3
	segmen 1										segmen 2		

Gambar 4.14. Contoh chromosome untuk mTSP

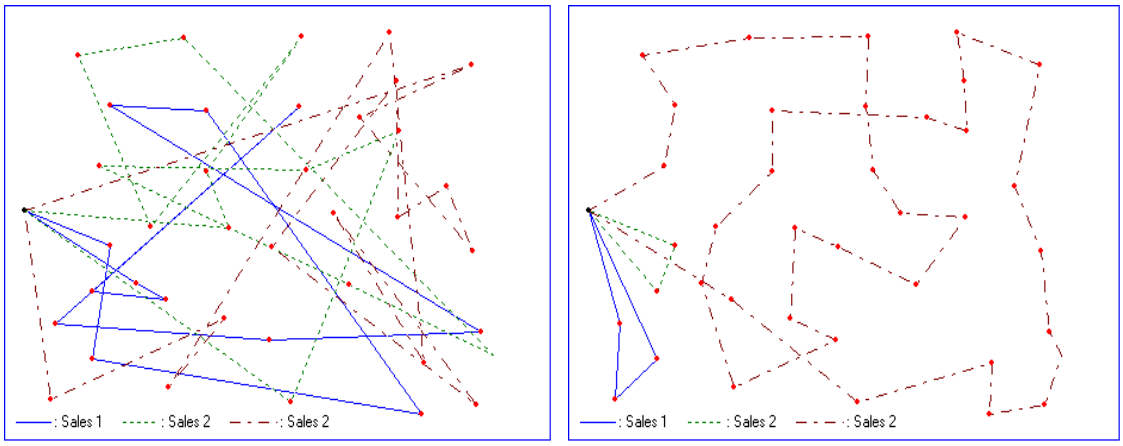
Gen-gen yang ada pada segmen 1 (posisi 1 sampai 10) menunjukkan urutan daerah yang dikunjungi, segmen 2 (posisi 11 sampai 13) menunjukkan banyaknya daerah yang dikunjungi tiap salesman. Dengan asumsi bahwa tiap salesman berangkat dari kantor pusat (KP) dan harus kembali juga ke kantor pusat, maka dari Gambar 2.4 dihasilkan rute tiap salesman sebagai berikut:

Salesman 1 : KP → daerah 2 → daerah 5 → daerah 1 → KP

Salesman 2 : KP → daerah 7→ daerah 6 → daerah 10 → daerah 4 → KP

Salesman 3 : KP → daerah 9 → daerah 8 → daerah 3 → KP

Untuk sebuah kasus dengan 40 node, Mahmudy (2008a) menunjukkan solusi yang dihasilkan oleh GAs seperti pada Gambar 4.15.



Generasi 1
Total jarak = 6884,3

Generasi 50000
Total jarak = 2703

Gambar 4.15. Contoh solusi mTSP untuk 40 node

mTSP telah diterapkan misalnya untuk kasus distribusi air mineral (Sari, RN & Mahmudy 2015).

4.9. Vehicle Routing Problem With Time Window (VRPTW)

Vehicle Routing Problem (VRP) merupakan permasalahan optimasi yang ada pada berbagai sistem distribusi. Pada permasalahan ini terdapat sejumlah kendaraan yang harus melayani sejumlah pelanggan. Setiap kendaraan punya kapasitas angkut tertentu dan setiap pelanggan mempunyai kuantitas permintaan tertentu. Solusi yang ingin dicapai adalah bagaimana memenuhi semua permintaan pelanggan dengan biaya minimum atau total jarak perjalanan terpendek.

Salah satu variasi dari VRP adalah *Vehicle Routing Problem with Time Windows* (VRPTW). Pada VRPTW terdapat *constraint* tambahan yaitu setiap pelanggan mempunyai waktu ketersediaan tertentu. Mereka hanya bisa dilayani pada interval waktu tersebut. Jika kendaraan datang sebelum waktu tersebut maka kendaraan harus menunggu. Sebaliknya jika kendaraan datang setelah waktu tersebut maka artinya pelanggan tersebut tidak akan dilayani. *Constraint* ini disebut *time window*.

VRPTW telah diterapkan pada berbagai kasus distribusi, misalnya:

- Distribusi makanan instan (Saputri, Mahmudy & Ratnawati 2015).
- Distribusi minuman (Harun, Mahmudy & Yudistira 2014).
- Distribusi beras bersubsidi (Putri, FB, Mahmudy & Ratnawati 2015).

Secara umum VRPTW bisa diselesaikan oleh algoritma evolusi dengan menggunakan representasi kromosom permutasi.

4.10. Rangkuman

Pada bab ini telah dibahas penerapan berbagai representasi chromosome dan operator reproduksi untuk menyelesaikan berbagai permasalahan kombinatorial. Representasi permutasi bisa digunakan untuk menyatakan sebuah solusi pada *Travelling Salesman Problem* (TSP). Setiap gen pada chromosome berupa angka integer yang menyatakan nomer dari tiap simpul. Metode crossover yang bisa digunakan adalah dengan melakukan modifikasi *one-cut-point crossover* yang digunakan pada representasi biner. Metode mutasi yang dibahas adalah *reciprocal exchange mutation* dan *insertion mutation*.

Multi Travelling Salesman Problem (m-TSP) merupakan pengembangan dari TSP. Pada permasalahan ini terdapat lebih dari seorang *salesman.*

Persoalan transportasi berkaitan dengan pendistribusian suatu komoditas atau produk dari sejumlah sumber (*supply*) kepada sejumlah tujuan (*destination*, *demand*) dengan tujuan meminimumkan ongkos pengangkutan yang terjadi.

Flow-Shop Scheduling Problem (FSP) berkaitan dengan penjadwalan sejumlah *j* job pada sejumlah *m* mesin. Semua job mempunyai urutan pemrosesan (operasi) yang sama. Representasi permutasi seperti yang digunakan pada TSP bisa diadopsi untuk FSP. Setiap gen pada chromosome menyatakan nomer dari tiap job. Operator crossover dan mutasi yang sama pada TSP juga bisa digunakan.

Two-stage Assembly Flowshop merupakan variasi dari FSP. Pada permasalahan ini sebuah job memiliki *m* operasi yang bisa dikerjakan pada *m* mesin secara paralel.

Pemrosesan tahap kedua (*assembly stage*) hanya bisa dilakukan setelah semua operasi pada tahap pertama telah diselesaikan. Representasi permutasi seperti yang digunakan pada TSP bisa diadopsi untuk *two-stage assembly flowshop*.

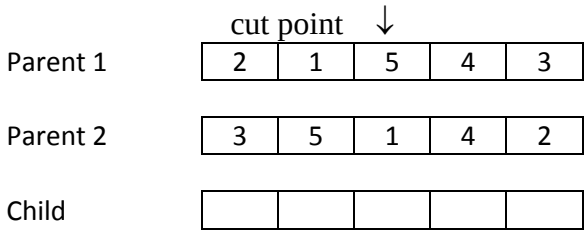
Job-Shop Scheduling Problem (JSP) merupakan perluasan dari FSP. Pada masalah ini tiap job mungkin mempunyai urutan operasi yang berbeda. Pada beberapa kasus, sebagian job hanya memerlukan sebagian mesin. Representasi permutasi seperti yang digunakan pada TSP bisa diadopsi untuk JSP.

Flexible job-shop problem (FJSP) merupakan bentuk umum (*generalized form*) dari JSP klasik. Pada permasalahan FSP, sebuah job memiliki beberapa operasi. Sebuah operasi bisa dikerjakan pada beberapa pilihan mesin. Skenario ini lebih dekat dengan kasus nyata yang ditemui pada industri manufaktur.

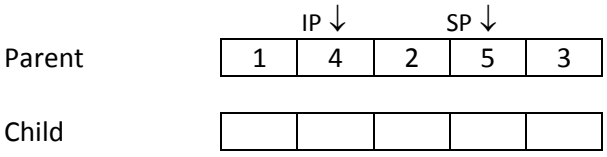
Bab ini juga menyajikan beberapa alternatif representasi chromosome untuk tiap permasalahan.

4.11. Latihan

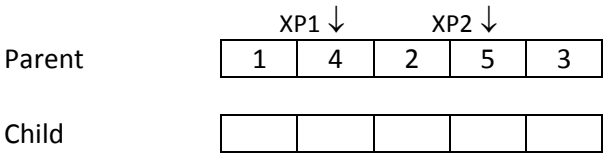
1. Jelaskan karakteristik dari masalah kombinatorial?
2. Untuk studi kasus TSP pada Sub-Bab 4.2 terdapat chromosome *P*=[3 4 2 1 5]. Hitung nilai total jarak dan fitnessnya!
3. Tentukan chromosome child untuk crossover pada representasi permutasi berikut!



4. Tentukan chromosome child untuk *insertion mutation* pada representasi permutasi berikut!



5. Tentukan chromosome child untuk *reciprocal exchange mutation* pada representasi permutasi berikut!



6. Untuk dua individu pada permasalahan transportasi berikut tentukan offspring yang terbentuk dari proses crossover!

$$P_1 = \begin{bmatrix} 0 & 0 & 0 & 10 \\ 10 & 5 & 0 & 0 \\ 0 & 0 & 5 & 0 \end{bmatrix}$$

$$P_2 = \begin{bmatrix} 5 & 0 & 0 & 5 \\ 0 & 5 & 5 & 5 \\ 5 & 0 & 0 & 0 \end{bmatrix}$$

7. Untuk individu pada permasalahan transportasi berikut tentukan offspring yang terbentuk dari proses mutasi dengan menggunakan titik sudut yang diberi warna kuning!

$$P = \begin{bmatrix} 0 & 0 & 0 & 10 \\ 10 & 5 & 0 & 0 \\ 0 & 0 & 5 & 0 \end{bmatrix}$$

8. Konversikan representasi permutasi *P*=[2 1 5 12 7 9 4 10 8 3 6 11] untuk menjadi solusi permasalah transportasi!
9. Hitung makespan untuk solusi FJSP dari representasi real *P*=[57 142 78 152 121 241 17 79 213 7] pada kasus di Sub-Bab 4.7!
10. Hitung makespan untuk solusi FJSP dari representasi real *P*=[129 114 147 186 220 58 159 11 31 235] pada kasus di Sub-Bab 4.7!

BAB 5

Topik Lanjut pada Algoritma Genetika

5.1. Pengantar

Meskipun GAs dianggap *powerful* untuk menyelesaikan berbagai permasalahan rumit, implementasi GAs sederhana seringkali tidak cukup efektif untuk menyelesaikan permasalahan kompleks dengan area pencarian yang sangat luas. Representasi chromosome dan operator genetika yang tepat, kombinasi (*hybrid*) dengan metode lain, dan strategi yang efisien untuk menghindari konvergensi dini diperlukan untuk memperkuat kemampuan GAs (Lozano & Herrera 2003; Rothlauf 2006). Bab ini membahas bagaimana strategi-strategi ini diterapkan.

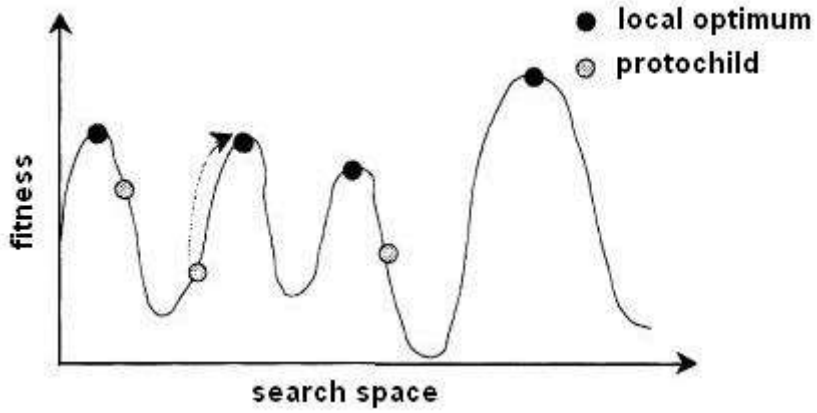
5.2. Hybrid Genetic Algorithms (HGAs)

Algoritma genetika murni memberikan hasil kurang optimum pada ruang pencarian yang kompleks. Penggabungan (*hybridisation*) dengan teknik lain dapat meningkatkan akurasi dan efisiensi pencarian solusi optimum (Mahmudy, Marian & Luong 2014). Hibridisasi GAs dengan teknik pencarian lokal (*local search* / LS) menghasilkan *memetic algorithms* (MAs). Teknik LS sederhana yang bisa dipakai misalnya algoritma *hill-climbing* yang sukses digunakan pada optimasi fungsi tanpa kendala (Mahmudy 2008b).

Kekuatan utama MAs adalah keseimbangan antara kemampuan eksplorasi GAs dalam pencarian pada area global dan kemampuan eksplotasi LS dalam area local (Lozano et al. 2004). Dalam implementasinya, LS diterapkan pada setiap individu baru dengan menggerakkannya menuju optimum lokal sebelum dimasukkan ke dalam populasi. Dengan mengacu struktur GAs murni pada Sub-Bab 2.2 maka struktur MAs bisa disusun dengan menambahkan perbaikan lokal sebagai berikut:

```
procedure AlgoritmaGenetika
begin
  t = 0
  inisialisasi P(t)
  while (bukan kondisi berhenti) do
    reproduksi C(t) dari P(t)
    evaluasi P(t) dan C(t)
    perbaiki C(t)
    seleksi P(t+1) dari P(t) dan C(t)
    t = t + 1
  end while
end
```

Mekanisme perbaikan offspring tersibut diilustrasikan pada Gambar 5.1. Anak yang baru terbentuk (*protochild*) akan didorong menuju optimum lokal.



Gambar 5.1. MAs dan optimasi lokal (Gen & Cheng 2000)

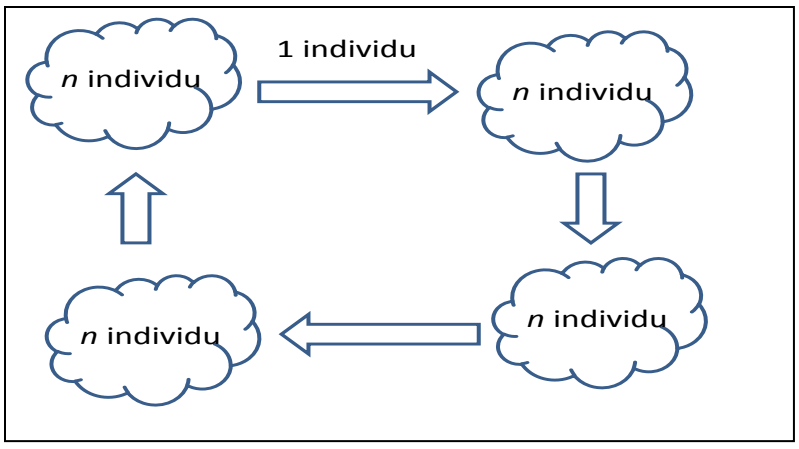
Karena GAs bisa dihibridisasi dengan algoritma meta-heuristik yang lain (tidak selalu LS) maka dalam pembahasan selanjutnya disebut *hybrid* GAs (HGAs). Salah satu penerapannya misalnya hibridisasi *real-coded* GAs (RCGA) dengan *variable neighbourhoods search* (VNS) untuk penyelesaian permasalahan *part type selection* dan *machine loading* pada *flexible manufacturing system* (FMS) (Mahmudy, Marian & Luong 2013c). Penyelesaian kedua permasalahan tersebut secara terintegrasi dikenal sangat sulit sehingga GAs dihibridisasi dengan algoritma lainnya seperti *particle swarm optimization* (Biswas & Mahapatra 2008) dan *simulated annealing* (Yogeswaran, Ponnambalam & Tiwari 2009)

5.3. Parallel Genetic Algorithms (PGAs)

Pada ruang pencarian yang luas dan kompleks, GAs sering terjebak dalam daerah optimum lokal. Hal ini terjadi karena kurangnya keragaman individu dalam populasi. Hal ini bisa diatasi dengan meletakkan individu-individu dalam beberapa sub-populasi. Pada tiap sub-populasi diterapkan operator genetika (crossover, mutasi, dan seleksi) yang berbeda. Operator migrasi digunakan untuk memindahkan satu atau beberapa individu dari satu sub-populasi ke dalam sub-populasi lain. Pendekatan ini menghasilkan metode yang disebut algoritma genetika terdistribusi (*distributed genetic algorithms*, DGAs). DGAs terbukti efektif menjaga keragaman populasi dan meningkatkan kualitas hasil pencarian (Mahmudy 2009).

Individu-individu yang ada juga bisa diletakkan pada beberapa sub-populasi yang diproses pada beberapa komputer secara paralel. Hal ini untuk mengurangi waktu komputasi pada masalah yang sangat kompleks (Defersha & Chen 2010; Qi, Burns & Harrison 2000). Karena itu DGAs sering juga disebut *parallel genetic algorithms* (PGAs).

Mekanisme migrasi sederhana dideskripsikan pada Gambar 5.2 yang menunjukkan ada empat sub-populasi. Pada setiap *g* generasi, satu invidu terbaik dipindahkan ke sub-populasi yang lain.



Gambar 5.2. Mekanisme migrasi

5.4. Nilai Parameter Adaptif

Kinerja GAs ditentukan oleh kemampuannya dalam menjelajahi (*explore*) dan mengeksploitasi (*exploit*) ruang pencarian (*search space*). Eksplorasi merujuk pada kemampuan untuk menginvestigasi area baru pada ruang pencarian. Eksploitasi merujuk pada kemampuan untuk meningkatkan kualitas solusi pada area tetangga (*neighbourhoods*) dari solusi yang didapatkan melalui eksplorasi (Lozano & Herrera 2003). Dari sini bisa disimpulkan bahwa keseimbangan kemampuan eksplorasi dan eksploitasi sangat penting untuk mendapatkan solusi yang baik (optimum atau mendekati optimum).

Keseimbangan kemampuan eksplorasi dan eksploitasi bisa didapatkan melalui penentuan *crossover rate* dan *mutation rate* yang tepat (Lozano & Herrera 2003). Jika menggunakan nilai *crossover rate* yang terlalu rendah maka GAs akan sangat tergantung pada proses mutasi. Walaupun *mutation rate* yang tinggi memungkinkan GAs mempunyai level eksplorasi dan diversitas populasi yang tinggi, *crossover rate* yang rendah membuatnya tidak bisa secara efektif 'belajar' dari generasi sebelumnya. Hal ini menyebabkan ruang pencarian tidak bisa diekploitasi secara efektif (Mahmudy 2014a).

Hal sebaliknya terjadi jika *crossover rate* yang tinggi dan *mutation rate* yang rendah digunakan. GAs akan mengalami penurunan kemampuan untuk menjaga diversitas pupolasi. *Crossover rate* yang tinggi akan menghasilkan *offspring* yang mempunyai kemiripan yang tinggi dengan induknya. Hal ini menyebabkan GAs mengalami konvergensi dini hanya dalam beberapa generasi dan kehilangan kesempatan untuk mengeksplorasi area lain dalam ruang pencarian (Mahmudy 2014a).

Penentuan kombinasi terbaik *crossover rate* dan *mutation rate* merupakan pekerjaan sulit dan memerlukan beberapa percobaan pendahuluan (Mahmudy, Marian & Luong 2013e). Untuk permasalahan yang berbeda dibutuhkan nilai yang berbeda pula. Karena itu beberapa penelitian menerapkan mekanisme perubahan *crossover rate* dan *mutation rate* secara adaptif sepanjang generasi (Im & Lee 2008; Liqun et al. 2010; Mahmudy & Rahman 2011). Serangkaian percobaan menunjukkan bahwa penggunaan

tingkat reproduksi adaptif mempercepat pergerakan GA ke daerah *feasible* yang sekaligus mempercepat pencapaian solusi (Mahmudy & Rahman 2011).

Mahmudy and Rahman (2011) menerapkan pengaturan *mutation rate* secara adaptif sepanjang generasi. Pada setiap generasi dihitung rata-rata nilai fitness seluruh individu dalam populasi (*fAvg*). Jika ada peningkatan rata-rata nilai fitness yang signifikan dibanding generasi sebelumnya (*fAvg>>fAvgOld*) maka nilai *mutation rate* diturunkan. Hal ini memungkinkan GAs untuk lebih fokus mengeksploitasi ruang pencarian lokal. Jika terjadi hal yang sebaliknya (tidak ada peningkatan signifikan) maka nilai *mutation rate* dinaikkan. Hal ini memungkinkan GA untuk lebih memperluas pencarian (eksplorasi) dengan melompati daerah optimum lokal. *Pseudo-code* dari mekanisme ini disajikan pada Gambar 5.3.

```
PROCEDURE UpdateMutationRate
Input:
     fAvg:      rata-rata nilai fitness pada generasi sekarang (t)
     fAvgOld:   rata-rata nilai fitness pada generasi sebelumnya (t-
                1)
     threshold: nilai perbedaan yang menyatakan bahwa ada peningkatan
                yang signifikan
     mutRate:   nilai mutation yang belum berubah

Output:
     mutRate:   nilai mutation yang telah berubah

if fAvg- fAvgOld>threshold then
   mutRate ← mutRate * 0.95
else
   mutRate ← mutRate * 1.1;
endif

if mutRate>mutRateMax then
   mutRate ← mutRateMax
else if mutRate<mutRateMin then
   mutRate ← mutRateMin
endif

END PROCEDURE
```

Gambar 5.3. Mekanisme pengubahan mutation rate secara adaptif

Pada pseudo-code di atas ditambahkan satu mekanisme untuk menjaga nilai *mutation rate* dalam range [*mutRateMin*, *mutRateMax*].

5.5. Rangkuman

Pada bab ini telah dibahas tiga teknik lanjut yang bisa diterapkan untuk memperkuat kemampuan algoritma genetika klasik, yaitu: *Parallel Genetic Algorithms* (PGAs), pengaturan nilai parameter algoritma genetika secara adaptif, dan *Hybrid Genetic Algorithms* (HGAs).

5.6. Latihan

1. Apa tujuan dari penerapan algoritma genetika terdistribusi?
2. Jelaskan mekanisme kerja dari algoritma genetika terdistribusi!
3. Apa tujuan dari pengembangan *Hybrid Genetic Algorithms* (HGAs)?
4. Apa tujuan dari penerapan nilai parameter adaptif?

BAB 6

Evolution Strategies (ES)

6.1. Pengantar

Teknik optimasi *evolution strategies* (ES) dicetuskan sejak awal tahun 1960-an dan kemudian dikembangkan lebih lanjut pada tahun 1970-an oleh Ingo Rechenberg, Hans-Paul Schwefel, dan rekan-rekannya di Technical University of Berlin (TUB) (Beyer & Schwefel 2002). Seperti halnya GAs, ES telah diaplikasikan dalam berbagai bidang, misalnya penjadwalan pemrosesan sinyal digital pada system multiprocessor (Greenwood, G W, Gupta & McSweeney 1994), pemrosesan citra dan *computer vision* (Louchet 2000), optimasi pelepasan *airbag* secara otomatis pada mobil (Ostertag, Nock & Kiencke 1995), penjadwalan tugas pada *real-time distributed computing systems* (Greenwood, G. W., Lang & Hurley 1995), optimasi komposisi pakan ternak (Milah & Mahmudy 2015), dan penentuan rute distribusi produk (Harun, Mahmudy & Yudistira 2014; Munawaroh & Mahmudy 2015b; Vista & Mahmudy 2015). ES juga cukup efektif dikombinasikan/dihibridisasi dengan algoritma lain seperti *particle swarm optimization* untuk penjadwalan staff (Nissen & Günther 2009). ES juga diterapkan untuk mendapatkan model terbaik dari Fuzzy AHP (Putri, AMDA, Mahmudy & Cholissodin 2015).

Ciri utama *evolution strategies* (ES) adalah penggunaan vektor bilangan pecahan (*real-vector*) sebagai representasi solusi. Berbeda dengan GAs yang menggunakan crossover sebagai operator reproduksi utama dan mutasi sebagai operator penunjang, ES lebih bertumpu pada operator mutasi. Mekanisme *self-adaptation* digunakan untuk mengontrol perubahan nilai parameter pencarian. GAs dan ES bisa digunakan untuk menyelesaikan permasalahan yang sama. Tetapi mana yang terbaik di antara kedua metode tersebut sangat tergantung pada permasalahan yang dihadapi.

6.2. Struktur Dasar Evolution Strategies

Beberapa notasi digunakan oleh ES. μ (*miu*) menyatakan ukuran populasi (sama seperti *popSize* pada GAs). λ (*lambda*) menyatakan banyaknya offspring yang dihasilkan pada proses reproduksi (sama seperti *crossover rate* dan *mutation rate* pada GAs). Beberapa penelitian menyarankan besarnya nilai λ sebesar 7μ.

Apabila *P(t)* dan *C(t)* merupakan populasi (*parents)* dan *offspring* pada generasi ke-*t*, maka siklus ES dapat dideskripsikan sebagai berikut:

```
procedure EvolutionStrategies
begin
  t = 0
  inisialisasi P(t): generate μ individu
  while (bukan kondisi berhenti) do
    recombinasi: produksi C(t) sebanyak λ dari P(t)
    mutasi C(t)
    seleksi P(t+1) dari P(t) dan C(t)
    t = t + 1
  end while
end
```

Perhatikan bahwa siklus ini serupa dengan siklus algoritma genetika (GAs). Perbedaan nyata ES dan GAs adalah pada operator yang digunakan. Perbedaan yang lain adalah mutasi pada GAs digunakan untuk menghasilkan individu baru (*offspring*) sebagai tambahan dari *offspring* yang diproduksi oleh operator crossover. Pada ES, mutasi diterapkan pada *offspring* yang dihasilkan proses rekombinasi. Rekombinasi pada ES mirip dengan operator crossover pada GAs tapi bisa menggunakan lebih dari satu induk.

Karena ES lebih mengandalkan mutasi, maka proses rekombinasi tidak selalu digunakan. Secara umum terdapat empat tipe proses dari ES, yaitu:

- (μ, λ)
- (μ/r, λ)
- (μ + λ)
- (μ/r + λ)

ES(μ,λ) tidak menggunakan rekombinasi dalam proses reproduksi. Seleksi menggunakan *elitism selection* hanya melibatkan individu dalam *offspring*, individu induk dalam populasi tidak dilibatkan. ES(μ/r,λ) serupa dengan ES(μ,λ) dengan tambahan melibatkan proses rekombinasi. ES(μ+λ) tidak menggunakan rekombinasi dan proses seleksi menggunakan *elitism selection* melibatkan individu *offspring* dan induk.

6.3. Siklus ES (μ, λ)

Permasalahan pada Sub-Bab 2.4 (Studi Kasus: Maksimasi Fungsi dengan Presisi Tertentu) akan digunakan untuk menjelaskan siklus ES secara detil.

6.3.1. Representasi Chromosome

Seperti halnya untuk real-coded GA pada pada Bab 3, variabel keputusan (x_1 dan x_2) langsung menjadi gen string *chromosome*. Selain gen yang menyatakan variabel keputusan, parameter tambahan yang melekat pada setiap *chromosome* adalah σ (*sigma*). Nilai ini menyatakan level mutasi untuk *chromosome* tersebut. Nilai ini akan ikut berubah secara adaptif sepanjang generasi. Jika *P* adalah satu *chromosome* maka $P=(x_1,x_2,\sigma_1,\sigma_2)$ dengan panjang string sebesar 4.

6.3.2. Inisialisasi

Populasi inisial dibangkitkan secara random. Nilai x_1 dan x_2 dibangkitkan dalam rentang variabel ini (lihat Modul 1). Nilai σ_1 dan σ_2 dibangkitkan dalam rentang [0,1]. Misalkan ditentukan μ=4 maka akan dihasilkan populasi seperti contoh berikut:

$P(t)$	x_1	x_2	σ_1	σ_2	$f(x_1,x_2)$
P_1	1,48980	2,09440	0,14197	0,91090	19,8212830
P_2	8,49170	2,57540	0,53801	0,86904	34,7060873
P_3	-1,84610	1,70970	0,99835	0,49351	11,5863900
P_4	5,81140	5,07790	0,40521	0,98911	14,5620828

6.3.3. Reproduksi

Karena rekombinasi tidak digunakan maka hanya mutasi yang berperan menghasilkan *offspring*. Misalkan $P=(x_1,x_2,\sigma_1,\sigma_2)$ adalah individu yang terpilih untuk melakukan mutasi, maka dihasilkan offspring $P'=(x'_1,x'_2,\sigma'_1,\sigma'_2)$ sebagai berikut:

$$x' = x + \sigma\, N(0,1)$$

Rumusan ini bisa didetailkan sebagai berikut:

$$x'_1 = x_1 + \sigma_1\, N(0,1)$$

$$x'_2 = x_2 + \sigma_2\, N(0,1)$$

$N(0,1)$ merupakan bilangan acak yang mengikuti sebaran normal dengan rata-rata sebesar 0 dan standard deviasi sebesar 1. Pada program komputer, nilai $N(0,1)$ bisa didapatkan dengan membangkitkan dua bilangan random r_1 dan r_2 pada interval [0,1]. Rumus yang digunakan adalah (Schwefel 1995):

$$N(0,1) = \sqrt{-2.\ln r_1}\,\sin 2\pi r_2$$

Misalkan r_1 = 0,4749 dan r_2 = 0,3296 maka didapatkan $N(0,1)$= 1,0709.

Nilai σ dinaikkan jika ada paling sedikit 20% hasil mutasi yang menghasilkan individu yang lebih baik dari induknya. Jika tidak maka nilai σ diturunkan. Misalkan λ=3×μ=12, maka setiap individu dalam populasi akan menghasilkan 3 *offspring*. Pada kasus ini, nilai σ akan dinaikkan jika ada setidaknya 1 offspring yang lebih baik.

Contoh hasil mutasi diberikan sebagai berikut:

$C(t)$	Induk	$N_1(0,1)$	$N_2(0,1)$	x'_1	x'_2	σ'_1	σ'_2	$f(x_1,x_2)$
C_1	P_1	0,0098	-0,8394	1,491191	1,3298	0,15617	1,00199	10,04952
C_2		1,0334	-0,6351	1,63651	1,5159	0,15617	1,00199	9,03814
C_3		-1,9967	-1,8970	1,206331	0,3664	0,15617	1,00199	27,06928
C_4	P_2	-0,0398	0,6565	8,470287	3,1459	0,48421	0,78214	24,40017
C_5		-0,7821	-0,2305	8,070926	2,3751	0,48421	0,78214	27,82881
C_6		1,3563	0,1430	9,221397	2,6997	0,48421	0,78214	19,00143
C_7	P_3	-1,1466	1,3203	-2,9908	2,3613	1,09818	0,54286	26,01116
C_8		1,1021	-1,9381	-0,74582	0,7532	1,09818	0,54286	26,00610

C_9		0,7094	-0,2813	-1,13787	1,5709	1,09818	0,54286	10,30137
C_{10}	P_4	1,8635	-0,2293	6,566503	4,8511	0,44573	1,08802	27,74543
C_{11}		0,5542	-1,2182	6,035966	3,873	0,44573	1,08802	17,29977
C_{12}		1,4608	-0,5120	6,403326	4,5715	0,44573	1,08802	30,40250

Perhatikan dari hasil mutasi ini, nilai σ' dari offspring yang dihasilkan P_1, P_3, dan P_4 dinaikkan dengan rumusan $\sigma' = \sigma \times 1{,}1$. Nilai σ' dari offspring yang dihasilkan P_2 diturunkan dengan rumusan $\sigma' = \sigma \times 0{,}9$.

6.3.4. Seleksi

Seleksi menggunakan *elitism selection* hanya melibatkan individu dalam *offspring*, individu induk dalam populasi tidak dilibatkan. Dari proses ini didapatkan populasi baru sebagai berikut:

$P(t+1)$	asal	x_1	x_2	σ_1	σ_2	$f(x_1,x_2)$
P_1	C_{12}	6,40333	4,57148	0,44573	1,08802	30,4025035
P_2	C_5	8,07093	2,37509	0,48421	0,78214	27,8288128
P_3	C_{10}	6,56650	4,85110	0,44573	1,08802	27,7454295
P_4	C_3	1,20633	0,36642	0,15617	1,00199	27,0692849

6.4. Siklus ES (μ/r + λ)

Pada sub-bab ini, siklus ES dibahas lagi dengan melibatkan proses rekombinasi. Populasi inisial pada Sub-Bab 6.3.2 digunakan lagi.

6.4.1. Reproduksi: Recombinasi dan Mutasi

Recombinasi dilakukan untuk menghasilkan offspring sebanyak λ dari sejumlah μ individu dalam populasi. Setiap satu individu offspring dihasilkan dari beberapa induk. Induk dipilih secara acak dari populasi. Metode rekombinasi paling sederhana adalah dengan menghitung rata-rata nilai elemen induk. Contoh proses rekombinasi diberikan sebagai berikut:

- Misalkan offspring didapatkan dari 2 induk. Jika P_1 dan P_3 terpilih maka akan didapatkan offspring C=(-0,17815, 1,90205, 0,57016, 0,70221).

- Misalkan offspring didapatkan dari 3 induk. Jika P_1, P_2 dan P_3 terpilih maka akan didapatkan offspring C=(2,71180, 2,12650, 0.55944, 0,75782).

Pada studi kasus ini, misalkan λ=6 dan offspring didapatkan dari 2 induk. Contoh hasil rekombinasi diberikan sebagai berikut

C(t)	Induk	x_1	x_2	σ_1	σ_2	$f(x_1,x_2)$
C_1	P_1 dan P_3	-0,17815	1,90205	0,57016	0,70221	16,6418295
C_2	P_2 dan P_3	3,32280	2,14255	0,76818	0,68128	19,5813015
C_3	P_1 dan P_4	3,65060	3,58615	0,27359	0,95001	9,5700496
C_4	P_2 dan P_4	7,15155	3,82665	0,47161	0,92907	12,5240357
C_5	P_1 dan P_3	-0,17815	1,90205	0,57016	0,70221	16,6418295
C_6	P_3 dan P_4	1,98265	3,39380	0,70178	0,74131	12,6500683

Dengan cara yang sama pada sub-bab sebelumnya, mutasi dilakukan. nilai σ dinaikkan jika hasil mutasi lebih baik. Sebaliknya jika hasil mutasi lebih buruk maka nilai σ diturunkan. Berikut ini contoh hasil mutasi:

$C'(t)$	$N_1(0,1)$	$N_2(0,1)$	x'_1	x'_2	σ'_1	σ'_2	$f(x_1,x_2)$
C_1	0,1885	0,2747	-0,07068	2,094946	0,62717	0,77243	21,3386958
C_2	-1,7947	-0,1359	1,944154	2,049965	0,84499	0,74940	19,9034447
C_3	-0,5603	-1,8657	3,497309	1,813724	0,30095	1,04501	10,9809788
C_4	0,6189	-0,4613	7,443427	3,398068	0,42445	0,83617	5,4075091
C_5	-0,1371	-0,4201	-0,25632	1,607053	0,51314	0,63199	11,2620580
C_6	1,1125	-0,2153	2,763377	3,234196	0,77195	0,81544	16,3293685

6.4.2. Seleksi

Seleksi menggunakan *elitism selection* melibatkan individu dalam *offspring* dan individu induk dalam populasi. Dari proses ini didapatkan populasi baru sebagai berikut

P(t+1)	asal	x_1	x_2	σ_1	σ_2	$f(x_1,x_2)$
P_1	P_2	8,49170	2,57540	0,53801	0,86904	34,7060873
P_2	C_1	-0,07068	2,09495	0,62717	0,77243	21,3386958
P_3	C_2	1,94415	2,04996	0,84499	0,74940	19,9034447
P_4	P_1	1,48980	2,09440	0,14197	0,91090	19,8212830

6.5. Studi Kasus ES (μ + λ): Optimasi Fungsi Berkendala

Perhatikan permasalahan pada sebuah perusahaan yang akan memproduksi dua jenis lemari, sebut saja lemari A dan lemari B. Untuk memproduksi kedua lemari tersebut dibutuhkan tiga macam bahan baku, yaitu: kayu, aluminium, dan kaca. Kebutuhan detil tiga bahan baku tersebut (dalam unit tertentu) untuk tiap buah lemari ditampilkan pada tabel berikut:

lemari	kayu	aluminium	kaca
A	10	9	12
B	20	8	18

Persedian bahan baku kayu, aluminium, dan kaca di gudang berturut-turut adalah 350, 200, dan 300. Jika keuntungan penjualan sebuah lemari A sebesar 400 dan B sebesar 500, berapakah banyaknya lemari A dan B harus diproduksi agar didapatkan keuntungan maksimum?

Untuk menyelesaikan permasalahan ini dibutuhkan sebuah model matematis. Model ini disusun atas sejumlah fungsi tujuan (*objective functions*) dan sejumlah kendala (*constraints*). Fungsi tujuan merepresentasikan tujuan yang ingin dioptimalkan (maksimumkan atau minimumkan). Jika banyaknya lemari yang harus diproduksi dilambangkan dengan x_1 dan x_2, maka fungsi tujuan bisa dinyatakan sebagai:

$$\text{maksimumkan} \quad f(x_1, x_2) = 400x_1 + 500x_2 \tag{6.1}$$

Kendala ketersediaan bahan baku bisa dinyatakan sebagai berikut:

$$\text{kendala 1:} \quad 10x_1 + 20x_2 \leq 350 \tag{6.2}$$

$$\text{kendala 2:} \quad 9x_1 + 8x_2 \leq 200 \tag{6.3}$$

$$\text{kendala 3:} \quad 12x_1 + 18x_2 \leq 300 \tag{6.4}$$

Pada optimasi fungsi berkendala, penentuan rumus perhitungan fitness harus dilakukan secara tepat agar solusi optimum bisa ditemukan secara efisien. Beberapa aturan diadopsi dari (Mahmudy & Rahman 2011) untuk menentukan mana individu yang lebih baik bisa dinyatakan sebagai berikut:

- Jika tidak ada kendala yang dilanggar maka sebuah individu dikatakan lebih baik dari individu yang lain jika nilai fungsi obyektifnya lebih besar (perhatikan ini berlaku untuk masalah maksimasi).
- Jika kedua individu melanggar minimal satu kendala maka dipilih yang total pelanggaran terhadap kendala lebih kecil. Hal ini untuk menjamin solusi yang dipilih memenuhi kendala sebanyak mungkin.

Berdasarkan dua aturan ini bisa disusun fungsi fitness sebagai berikut:

$$fitness(x_1, x_2) = f(x_1, x_2) - M(c_1 + c_2 + c_3) \quad (6.5)$$

M: bilangan positif yang cukup besar, misalnya pada kasus ini adalah 1000

$$c_1 = \begin{cases} 0, & \text{jika } 10x_1 + 20x_2 \leq 350 \\ (10x_1 + 20x_2) - 350, & \text{selainnya} \end{cases}$$

$$c_2 = \begin{cases} 0, & \text{jika } 9x_1 + 8x_2 \leq 200 \\ (9x_1 + 8x_2) - 200, & \text{selainnya} \end{cases}$$

$$c_3 = \begin{cases} 0, & \text{jika } 12x_1 + 18x_2 \leq 300 \\ (12x_1 + 18x_2) - 300, & \text{selainnya} \end{cases}$$

Contoh perhitungan fitness diberikan dalam tabel berikut

x_1	x_2	Round(x_1)	Round(x_2)	$f(x_1,x_2)$	c_1	c_2	c_3	fitness
21,9	0,3	22	0	8800	0	0	0	8800
0,2	15,9	0	16	8000	0	0	0	8000
10,3	11,4	10	11	9500	0	0	18	-8500
8,1	13,6	8	14	10200	10	0	48	-47800

Nilai x_1 dan x_2 merupakan bilangan pecahan (*real*). Karena permasalahan ini memerlukan solusi dalam bentuk bilangan bulat maka dalam perhitungan fitness nilai x_1 dan x_2 dibulatkan terlebih dahulu.

6.5.1. Inisialisasi

Populasi inisial dibangkitkan secara random. Nilai x_1 dan x_2 dibangkitkan sebagai bilangan pecahan (*real*) dalam rentang [0,50]. Misalkan ditentukan μ=4 maka akan dihasilkan populasi dengan contoh sebagai berikut:

P(t)	x_1	x_2	σ_1	σ_2	fitness
P_1	21,9	0,3	0,23241	0,02713	8800
P_2	0,2	15,9	0,82123	0,10383	8000
P_3	10,3	11,4	0,79231	0,93718	-8500
P_4	8,1	13,6	0,31982	0,75632	-47800

6.5.2. Reproduksi

Karena rekombinasi tidak digunakan maka hanya mutasi yang berperan menghasilkan *offspring*. Pada studi kasus ini, misalkan λ=2μ=8 maka dihasilkan offspring seperti contoh berikut:

C(t)	Induk	$N_1(0,1)$	$N_2(0,1)$	x'_1	x'_2	σ'_1	σ'_2	fitness
C_1	P_1	1,8023	-1,3414	22,3189	0,2636	0,20917	0,02442	8800
C_2		-0,7837	1,2838	21,7179	0,3348	0,20917	0,02442	8800
C_3	P_2	0,6234	-0,9298	0,7120	15,8035	0,68574	-1,02278	8400
C_4		-1,2394	-0,3293	-0,8178	15,8658	-1,36334	-0,36223	7600
C_5	P_3	1,0383	-2,0459	11,1227	9,4826	0,87154	1,03090	8900
C_6		1,9932	1,2427	11,8792	12,5646	0,87154	1,03090	-108700
C_7	P_4	-0,4513	-1,6313	7,9557	12,3662	0,35180	0,83195	-2800
C_8		-1,2093	2,0348	7,7132	15,1390	0,35180	0,83195	-85300

Perhatikan dari hasil mutasi ini, nilai σ' dari offspring yang dihasilkan P_2, P_3, dan P_5 dinaikkan dengan rumusan $\sigma' = \sigma \times 1{,}1$ karena paling tidak menghasilkan 1 anak yang lebih baik. Nilai σ' dari offspring yang dihasilkan P_1 diturunkan dengan rumusan $\sigma' = \sigma \times 0{,}9$.

6.5.3. Seleksi

Seleksi menggunakan *elitism selection* melibatkan individu dalam *offspring* dan individu induk dalam populasi. Dari proses ini didapatkan populasi baru sebagai berikut:

P(t+1)	asal	x_1	x_2	σ_1	σ_2	fitness
P_1	C_5	11,1227	9,4826	0,87154	1,03090	8900
P_2	P_1	21,9	0,3	0,23241	0,02713	8800
P_3	C_1	22,3189	0,2636	0,20917	0,02442	8800
P_4	C_2	21,7179	0,3348	0,20917	0,02442	8800

6.6. ES untuk Representasi Permutasi

Seperti telah diuraikan pada awal bab ini, ciri utama ES adalah penggunaan vektor bilangan pecahan (*real-vector*) sebagai representasi. Pada perkembangannya, ES juga diadopsi untuk permasalahan kombinatorial yang menggunakan representasi permutasi. Cara paling mudah adalah dengan menggunakan struktur ES yang hanya menggunakan mutasi tanpa rekombinasi. Mekanisme *self-adaptation* juga tidak digunakan. Pendekatan ini pada hakekatnya menghasilkan siklus yang sama seperti GAs tanpa crossover.

Adopsi mekanisme *self-adaptation* pada representasi permutasi bisa dilakukan dengan cara sederhana jika yang digunakan adalah *reciprocal exchange mutation* atau *insertion mutation*. Pada kasus ini, nilai σ menyatakan berapa kali proses *exchange* atau *insertion* dilakukan untuk menghasilkan satu anak. Misalkan diberikan contoh dua induk dalam tabel berikut:

***P*(*t*)**	***permutasi***	σ
P_1	[2 5 1 4 3]	1,3452
P_2	[4 1 5 3 2]	2,0728

Misalkan *reciprocal exchange mutation* digunakan dan $\lambda=3\mu$. Berdasarkan nilai σ yang dibulatkan, offspring dari P_1 dihasilkan melalui proses sekali pertukaran dan offspring dari P_2 dihasilkan melalui dua kali pertukaran. Contoh offspring yang dihasilkan ditampilkan dalam tabel berikut:

induk	**proses**	**offspring**
P_1 = [2 5 1 4 3]	tukar posisi 1 dan 3	C_1 = [**1** 5 **2** 4 3]
	tukar posisi 2 dan 5	C_2 = [2 **3** 1 4 **5**]
	tukar posisi 4 dan 5	C_3 = [2 5 1 **3** **4**]
P_2 = [4 1 5 3 2]	tukar posisi 2 dan 4 tukar posisi 1 dan 5	C_4 = [**2** **3** 5 **1** **4**]
	tukar posisi 1 dan 4 tukar posisi 3 dan 4	C_5 = [**3** 1 **4** **5** 2]
	tukar posisi 3 dan 4 tukar posisi 2 dan 4	C_6 = [4 **5** **3** **1** 2]

6.7. Rangkuman

Pada bab ini telah dibahas berbagai macam struktur ES beserta siklusnya. Struktur ES yang dibahas adalah ES(μ,λ), ES(μ/r,λ), ES(μ+λ), dan ES(μ/r+λ). Diuraikan juga perbedaan pokok ES dan GAs. Mekanisme *self-adaptation* untuk mengatur level mutasi dibahas dalam studi kasus.

6.8. Latihan

1. Jelaskan ciri utama *evolution strategies*!
2. Jelaskan perbedaan utama ES dengan GAs?
3. Misalkan terdapat himpunan individu dalam populasi dengan μ=4 dan λ=6 sebagai berikut:

individu	fitness
P_1	10
P_2	9
P_3	7
P_4	5

Terdapat juga himpunan *offspring* sebagai berikut:

individu	fitness
C_1	11
C_2	9
C_3	8
C_4	7
C_5	6
C_6	4

Tentukan himpunan individu yang lolos ke generasi selanjutnya pada ES(μ,λ)!

4. Kerjakan ulang Soal 3 untuk ES(μ+λ)!
5. Perhatikan fungsi berikut:

$$f(x_1, x_2) = x_1 \sin(2x_1) + x_2 \sin(4x_2), \quad 0 \le x_1 \le 10, 0 \le x_2 \le 10$$

Cari nilai minimum dari fungsi ini dengan menggunakan ES (μ/r+λ). Gunakan nilai μ=4 dan λ=8. Lakukan interasi sampai 3 putaran.

6. Untuk permasalahan pada Sub-Bab 6.5. Studi Kasus ES ($\mu + \lambda$): Optimasi Fungsi Berkendala, selesaikan dengan menggunakan ES (μ/r,λ). Gunakan nilai μ=4 dan λ=8. Lakukan interasi sampai 3 putaran.
7. Selesaikan persoalan transportasi pada Sub-Bab 4.6 dengan menggunakan ES (μ,λ). Gunakan nilai μ=4 dan λ=8. Lakukan interasi sampai 3 putaran.

BAB 7

Genetic Programming (GP) dan Evolutionary Programming (EP)

7.1. Genetic Programming

GP merupakan satu metode untuk secara sistematis mengevolusi program komputer. Ide dasarnya adalah bagaimana komputer dapat secara otomatis menghasilkan program komputer berdasarkan sebuah permasalahan yang diberikan. Berbeda dengan solusi GAs yang berupa nilai numeris berdasarkan fungsi obyektif yang digunakan, GP menghasilkan rangkaian program komputer yang bisa direpresentasikan dalam bentuk struktur pohon (*tree*).

GP telah diaplikasikan dalam berbagai bidang, misalnya untuk mendesain *analog circuit* pada skala industry secara otomatis (Koza et al. 2004), mendesain *quantum computing circuits* (Spector 2004), dan membantu manajer untuk mendesain organisasi yang optimal pada sebuah proyek (Khosraviani, Levitt & Koza 2004). GP juga cukup efektif dikombinasikan/dihibridisasi dengan algoritma lain seperti *ant colony optimisation* untuk memperbaiki desain geometris pada reflector *light-emitting diode* (LED). Perbaikan ini akan berpengaruh secara signifikan terhadap intensitas cahaya yang dipancarkan oleh LED (Hsu 2011). Pada bidang *data mining*, hybrid GP dan *simulated annealing* telah sukses diaplikasikan untuk membentuk pohon keputusan pada klasifikasi data (Folino, Pizzuti & Spezzano 2000).

Satu contoh permasalahan sederhana berdasarkan tabel berikut ini adalah 'susun sebuah fungsi dengan variabel bebas x_1 dan x_2 yang menghasilkan variabel tak bebas y'.

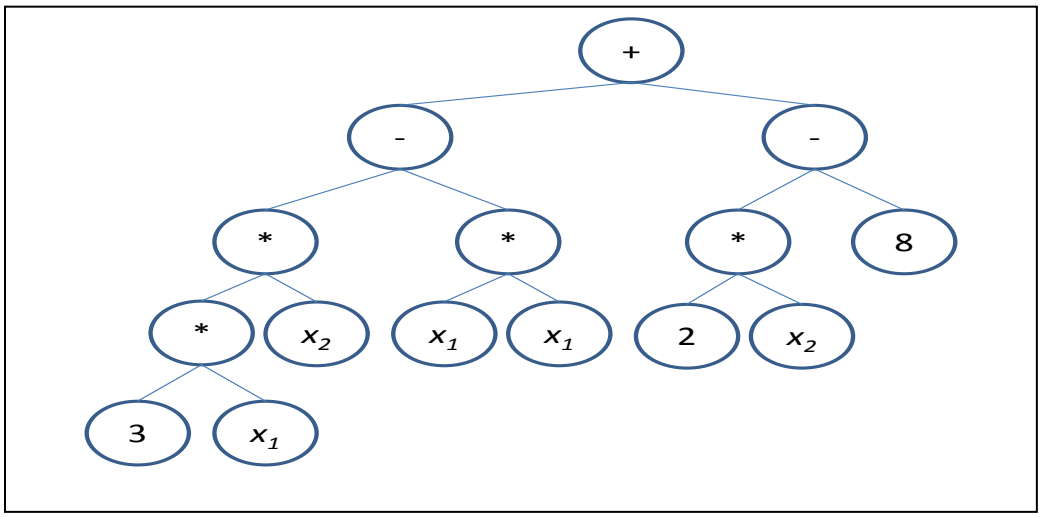
x_1	x_2	y
1	2	1
2	3	12
3	2	5
4	5	46

5	3	18
6	5	56
7	4	35
8	6	84
9	5	56
10	8	148

Tabel di atas sebenarnya dihasilkan dari sebuah fungsi non-linear dengan persamaan berikut:

$$y = f(x_1, x_2) = (3x_1x_2 - x_1^2) + (2x_2 - 8)$$

Fungsi ini bisa direpresentasikan dalam bentuk *binary tree* seperti ditunjukkan pada Gambar 7.1. Tree ini disusun atas sejumlah *node* dan *link*. Node menunjukkan instruksi yang harus dilaksanakan (misalnya * dan +). Link merujuk ke argumen dari tiap instruksi (misalnya x_1 dan x_2).



Gambar 7.1. Binary tree sebuah fungsi non-linear

Masalahnya adalah bagaimana merancang GP untuk menghasilkan *binary tree* di atas berdasarkan tabel yang diberikan (dalam hal ini fungsinya belum diketahui sebelumnya). Secara umum, siklus GP mirip dengan GAs. Dua hal yang harus ditambahkan pada representasi solusi GP adalah:

- Penentuan himpunan terminal (*set of terminals*)
- Penentuan himpunan fungsi (*set of functions*)

7.2. Siklus Genetic Programming

Untuk kasus penentuan fungsi non-liner bisa didefinisikan himpunan terminal dan fungsi sebagai berikut:

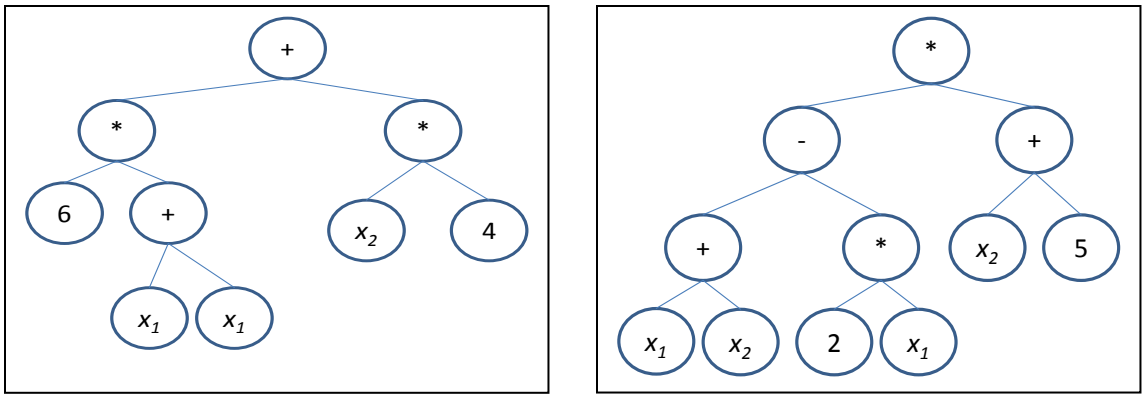
- Himpunan terminal: $T = \{x_1, x_2, x \in real\_number\}$
- Himpunan fungsi: $F = \{+, -, *, /\}$

7.2.1. Representasi Chromosome

Setiap chromosome berupa sebuah *binary tree*. Bagaimana sebuah *binary tree* disimpan/ditangani oleh program komputer, silahkan membuka ulang materi mata kuliah struktur data.

7.2.2. Inisialisasi dan Evaluasi

Chromosome dari dua individu yang dibangkitkan secara acak diberikan sebagai berikut:



individu 1 (P_1) individu 2 (P_2)

Gambar 7.2. Contoh dua individu random

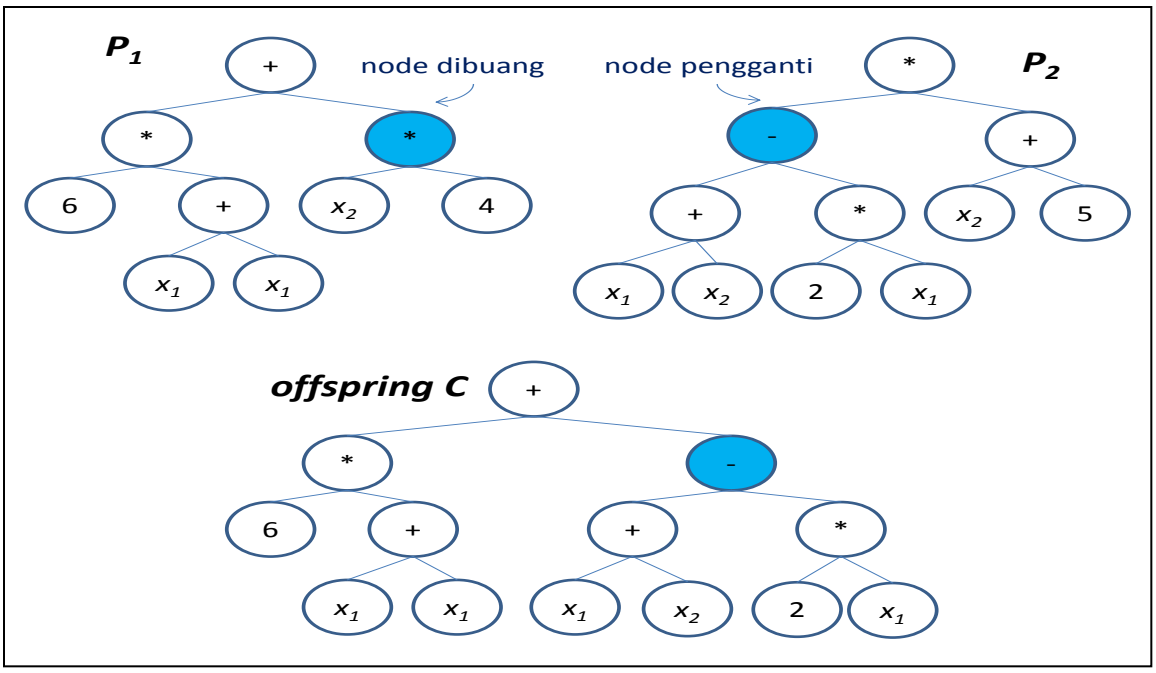
Evaluasi individu dilakukan dengan menyusun fungsi non-linear dari *binary tree* yang dihasilkan. Fitness dihitung berdasarkan total selisih antara output fungsi non-linear yang dihasilkan dengan nilai *y* pada tabel. Tabel berikut menunjukkan perhitungan total selisih output untuk P_1 dan P_2.

x_1	x_2	y	P_1 $y_{P_1} = 6(x_1 + x_1) + 4x_2$		P_2 $y_{P_2} = \big((x_1 + x_2) - 2x_1\big)(x_2 + 5)$	
			y_{P_1}	$\lvert y - y_{P_1} \rvert$	y_{P_2}	$\lvert y - y_{P_2} \rvert$
1	2	1	20	19	7	6
2	3	12	36	24	8	4
3	2	5	44	39	-7	12
4	5	46	68	22	10	36
5	3	18	72	54	-16	34
6	5	56	92	36	-10	66
7	4	35	100	65	-27	62
8	6	84	120	36	-22	106
9	5	56	128	72	-40	96
10	8	148	152	4	-26	174
		Total		371		596

Dari tabel di atas, fitness dari P_1 bisa dihitung sebagai $fitness = 1/(1 + 371) = 0{,}002688$. Fitness dari P_2 bisa dihitung sebagai $fitness = 1/(1 + 596) = 0{,}001675$. Hasil perhitungan ini menunjukkan bahwa P_1 lebih baik daripada P_2.

7.2.3. Crossover

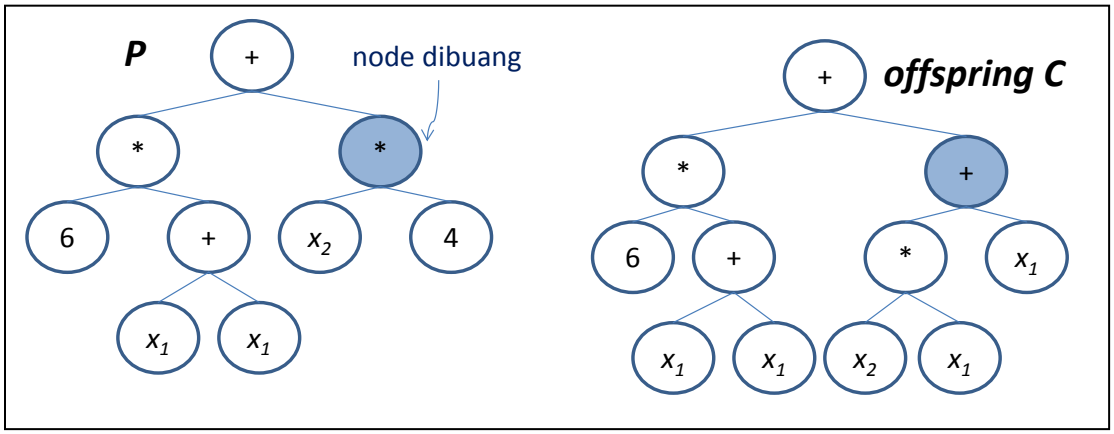
Crossover antara dua induk P_1 dan P_2 dilakukan dengan memilih satu node secara acak dari P_1. *Sub-tree* yang dimulai dari node ini dihapus dan digantikan dengan *sub-tree* yang dipilh secara acak dari P_2. Hasil *offspring C* yang terbentuk dari proses ini diilustrasikan dalam Gambar 7.3.



Gambar 7.3. Contoh proses crossover

7.2.4. Mutasi

Mutasi terhadap induk *P* dilakukan dengan memilih satu node secara acak. *Sub-tree* yang dimulai dari node ini digantikan dengan *sub-tree* yang dibangkitkan secara acak seperti pada proses inisialisasi. Hasil *offspring C* yang terbentuk dari proses ini diilustrasikan dalam Gambar 7.4.



Gambar 7.4. Contoh proses mutasi

7.2.5. Seleksi

Metode seleksi pada GP sama dengan yang digunakan pada GAs.

7.3. Evolutionary Programming (EP)

Evolutionary Programming (EP) mempunyai tujuan seperti GP untuk menghasilkan rangkaian program komputer tapi prinsip kerjanya seperti ES. Finite State Machines (FSM) biasanya digunakan untuk merepresentasikan program komputer.

Contoh sukses penerapan EP misalnya untuk mencari komposisi campuran minuman yang terbaik berdasarkan penilaian *tester*. Bahan campuran yang dipakai diantaranya: *sugar water, ginger flavoring, strawberry nectar, salt water, pineapple juice, tea, raspberry juice, peach nectar, milk, grapefruit juice, cranberry juice, coffee, apple juice, grape juice*, dan *chocolate*. Hasil percobaan yang didapatkan membuktikan kekuatan EP dalam pencarian solusi (Bell & Alexande 2007). Pada bidang kesehatan, EP digunakan untuk melatih sebuah *artificial neural network* (ANN) untuk mendeteksi penyakit kanker (Fogel et al. 1998).

EP juga digunakan dalam penjadwalan, misalkan pada *short-term hydrothermal scheduling* (Hota, Chakrabarti & Chattopadhyay 1999) dan penjadwalan proyek (Sebt & Alipouri 2013). EP juga cukup efektif dikombinasikan/dihibridisasi dengan algoritma lain seperti *particle swarm optimization* untuk penjadwalan perbaikan (*maintenance*) pada pembangkit tenaga listrik (Samuel & Rajan 2013).

7.4. Studi Kasus 1: Pohon Keputusan

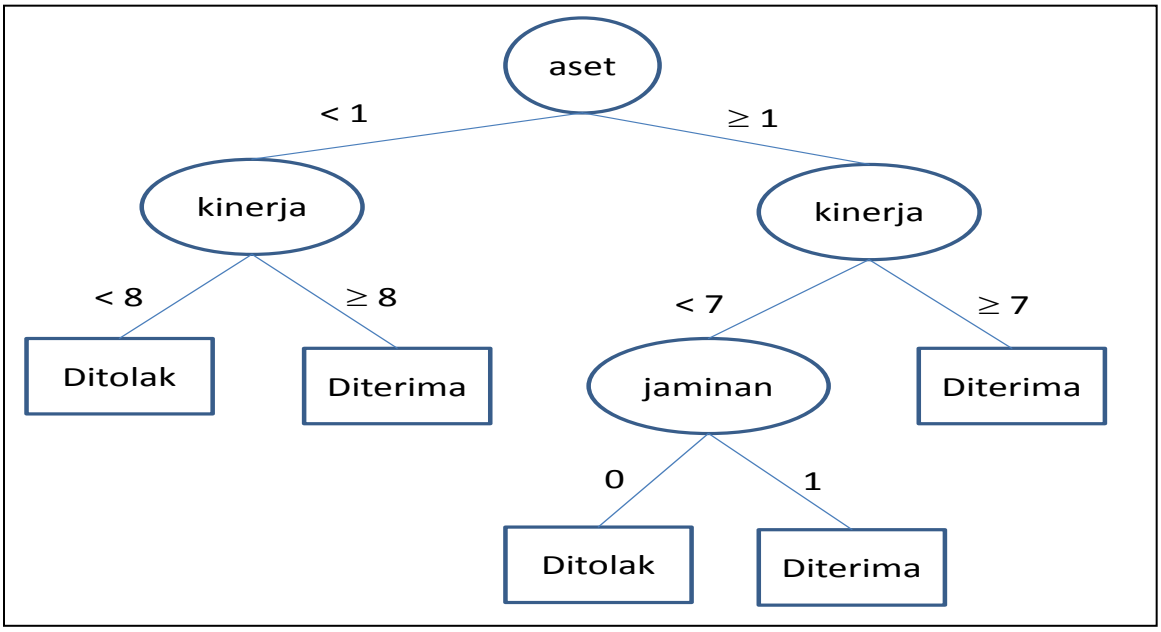
Perhatikan data historis keputusan kredit usaha yang diajukan oleh perusahaan pada sebuah bank sebagai berikut:

No	Aset (*A*)	Kinerja (*K*)	Jaminan (*J*)	Keputusan *(Kep*)
1	1.5	6	0	0
2	0.7	6	1	0
3	2	7	0	1
4	1.6	5	1	1
5	0.9	8	0	1
6	0.8	8	1	1

Data historis ini berdasarkan penilaian ahli perkreditan bank tersebut. Keputusan (*Kep*) untuk menerima (1) atau menolak (0) pengajuan kredit didasarkan atas tiga variabel yaitu: nilai aset (*A*) yang dipunyai perusahaan, nilai kinerja (*K*) perusahaan setahun terakhir, dan ketersediaan jaminan (*J*).

Bank memutuskan untuk membangun sistem yang bisa secara otomatis memberikan keputusan penerimaan. Sistem ini bisa memberikan output *Kep* jika misalkan diberikan input *A*=1,2, *K*=5, dan *J*=6.

Solusi GP berupa pohon keputusan (*decision tree*) seperti ditunjukkan pada Gambar 7.5. Gambar ini menunjukkan alur penerimaan/penolakan pengajuan kredit berdasarkan input *A*, *K*, dan *J*. Pengecekan dimulai dari node paling atas (*root*). Masalahnya adalah bagaimana menghasilkan pohon keputusan seperti ini berdasarkan data historis keputusan kredit usaha.



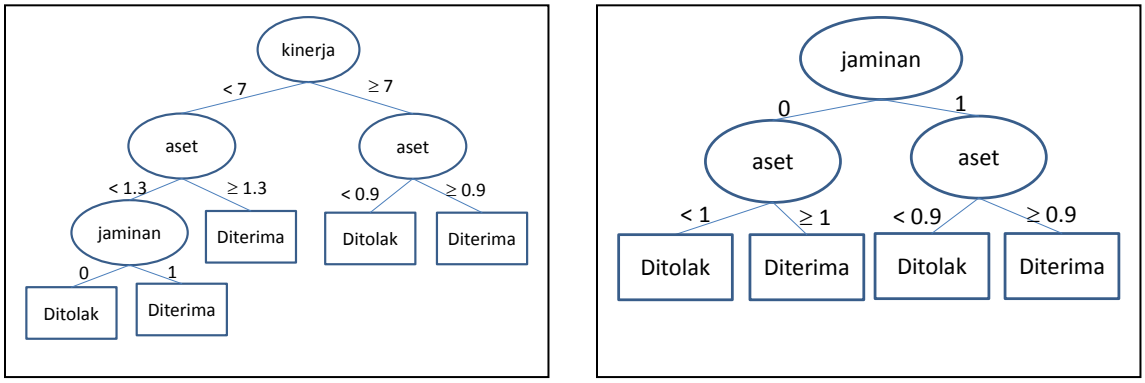
Gambar 7.5. Pohon keputusan penerimaan pengajuan kredit

7.4.1. Representasi Chromosome

Karena solusi yang diinginkan berupa pohon keputusan maka setiap chromosome berupa sebuah *binary tree*.

7.4.2. Inisialisasi dan Evaluasi

Chromosome dari dua individu yang dibangkitkan secara acak diberikan sebagai berikut:



individu 1 (P_1) individu 2 (P_2)

Gambar 7.6. Contoh dua individu random

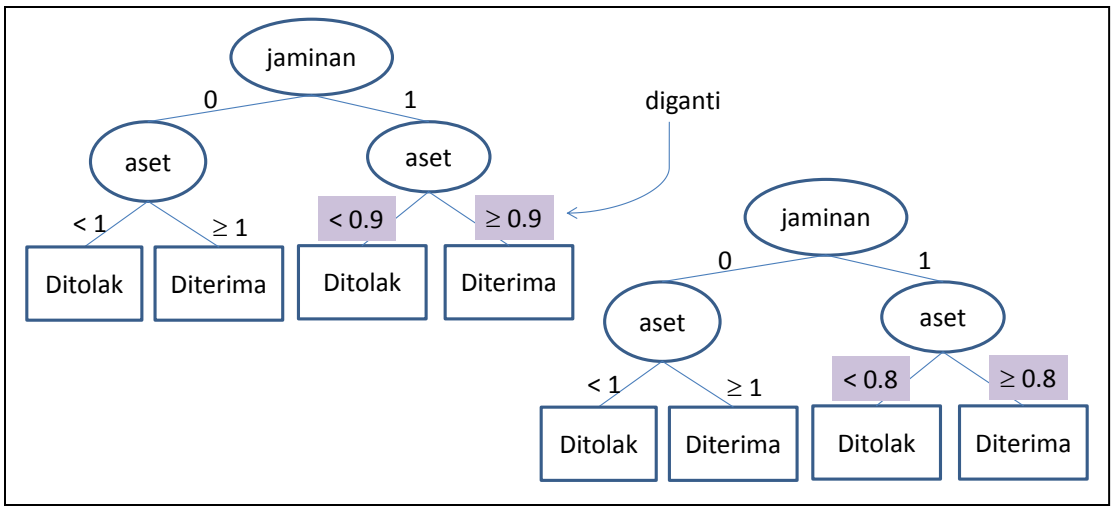
Evaluasi individu dilakukan dengan memasukkan data historis pada *tree* yang dihasilkan. Jika ada kesesuaian nilai keputusan (*K*) maka diberi skor 1. Karena ada 6 data historis maka fitness setiap individu berada pada range [0,6]. Perhatikan dua individu (P_1 dan P_2) ini kebetulan sama-sama mempunyai fitness sebesar 3.

No	Aset (*A*)	Kinerja (*K*)	Jaminan (*J*)	Keputusan (*Kep*)	individu P_1		individu P_2	
					Keputusan	Skor	Keputusan	Skor
1	1.5	6	0	0	1	0	1	0
2	0.7	6	1	0	1	0	0	1
3	2	7	0	1	1	1	1	1
4	1.6	5	1	1	1	1	1	1
5	0.9	8	0	1	1	1	0	0
6	0.8	8	1	1	0	0	0	0
Fitness						3		3

7.4.3. Crossover, Mutasi, dan Seleksi

Operator reproduksi dan seleksi sama seperti yang digunakan pada sub-bab sebelumnya.

Operator reproduksi dan seleksi sama seperti yang digunakan pada sub-bab sebelumnya. Khusus untuk mutasi, karena pada *link* terdapat angka yang menunjukkan batasan sebuah variabel maka harus ditambahkan mekanisme mutasi yang digunakan hanya untuk mengubah angka ini seperti ditunjukkan pada Gambar 7.7.



Gambar 7.7. Contoh mutasi mengubah angka

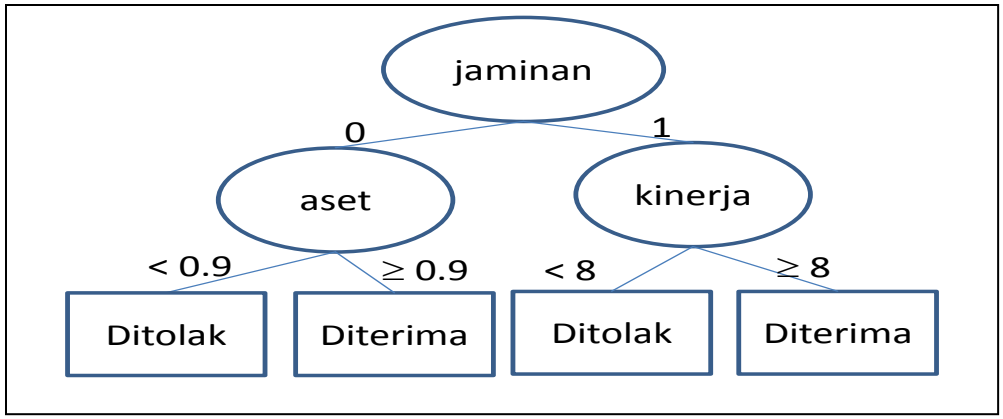
7.5. Rangkuman

Pada bab ini telah dibahas struktur GP dan EP beserta siklusnya. Pembahasan difokuskan pada bagaimana menggunakan GP dan EP untuk menghasilkan program komputer dalam bentuk *binary tree* dan *decision tree*.

7.6. Latihan

Untuk memperjelas pemahaman anda mengenai materi pada bab ini, kerjakanlah latihan berikut!

1. Jelaskan apakah output dari GP?
2. Apakah komponen utama pada representasi solusi dari GP?
3. Untuk kasus pada Sub-Bab 7.4, hitunglah fitness untuk chromosome *P* berikut:



4. Selesaikan studi kasus pada Sub-Bab 7.2. Siklus Genetic Programming sampai 5 generasi dengan menggunakan GP!
5. Selesaikan studi kasus pada Sub-Bab 7.4. Studi Kasus: Pohon Keputusan sampai 5 generasi dengan menggunakan GP!
6. Dengan menggunakan EP, susun *binary tree* untuk data pada tabel berikut!

x_1	x_2	y
3	4	18
4	2	24
6	3	51
6	4	48
7	1	87
8	2	104
9	3	123
10	4	144
10	6	128
11	9	143

Daftar Pustaka

1. Achnas, AH, Cholissodin, I & Mahmudy, WF 2015, 'Optimasi fuzzy inference system sugeno dengan algoritma hill climbing untuk penentuan harga jual rumah', *Journal of Environmental Engineering & Sustainable Technology*, vol. 2, no. 1, pp. 35-40.
2. Al-Hinai, N & ElMekkawy, T 2011, 'An efficient hybridized genetic algorithm architecture for the flexible job shop scheduling problem', *Flexible Services and Manufacturing Journal*, vol. 23, no. 1, pp. 64-85.
3. Allahverdi, A & Al-Anzi, FS 2008, 'The two-stage assembly flowshop scheduling problem with bicriteria of makespan and mean completion time', *Int J. Adv. Manuf. Technol*, vol. 37, pp. 166–177.
4. Azizah, EN, Cholissodin, I & Mahmudy, WF 2015, 'Optimasi fungsi keanggotaan fuzzy tsukamoto menggunakan algoritma genetika untuk penentuan harga jual rumah', *Journal of Environmental Engineering & Sustainable Technology*, vol. 2, no. 2, pp. 83-86.
5. Bell, C & Alexande, S 2007, *A Tasteful Example of Evolutionary Programming*, Southwestern University.
6. Beyer, H-G & Schwefel, H-P 2002, 'Evolution strategies – A comprehensive introduction', *Natural Computing*, vol. 1, no. 1, 2002/03/01, pp. 3-52.
7. Biswas, S & Mahapatra, S 2008, 'Modified particle swarm optimization for solving machine-loading problems in flexible manufacturing systems', *The International Journal of Advanced Manufacturing Technology*, vol. 39, no. 9, pp. 931-942.
8. Ciptayani, PI, Mahmudy, WF & Widodo, AW 2009, 'Penerapan algoritma genetika untuk kompresi citra fraktal', *Jurnal Ilmu Komputer*, vol. 2, no. 1, April, pp. 1-9.
9. Defersha, F & Chen, M 2010, 'A parallel genetic algorithm for a flexible job-shop scheduling problem with sequence dependent setups', *The International Journal of Advanced Manufacturing Technology*, vol. 49, no. 1, pp. 263-279.
10. Endarwati, DA, Mahmudy, WF & Ratnawati, DE 2014, 'Pencarian rute optimum dengan evolution strategies', *DORO: Repository Jurnal Mahasiswa PTIIK Universitas Brawijaya*, vol. 4, no. 10.
11. Fogel, DB, Wasson, EC, Boughton, EM & Porto, VW 1998, 'Evolving artificial neural networks for screening features from mammograms', *Artificial Intelligence in Medicine*, vol. 14, no. 3, pp. 317-326.
12. Folino, G, Pizzuti, C & Spezzano, G 2000, 'Genetic Programming and Simulated Annealing: A Hybrid Method to Evolve Decision Trees', in Poli, Ret al (eds), *Genetic Programming, European Conference, Edinburgh, Scotland, UK, April 15-16, 2000, Proceedings*, Springer, pp. 294-303.
13. Gen, M & Cheng, R 1997, *Genetic Algorithms and Engineering Design*, John Wiley & Sons, Inc., New York.
14. Gen, M & Cheng, R 2000, *Genetic Algorithms and Engineering Optimization*, John Wiley & Sons, Inc., New York.

15. Greenwood, GW, Gupta, A & McSweeney, K 1994, 'Scheduling tasks in multiprocessor systems using evolutionary strategies', *IEEE World Congress on Computational Intelligence, Proceedings of the First IEEE Conference on*, pp. 345 –349.
16. Greenwood, GW, Lang, C & Hurley, S 1995, 'Scheduling tasks in real-time systems using evolutionary strategies', *Parallel and Distributed Real-Time Systems, 1995. Proceedings of the Third Workshop on*, 25 Apr 1995, pp. 195-196.
17. Harun, IA, Mahmudy, WF & Yudistira, N 2014, 'Implementasi evolution strategies untuk penyelesaian vehicle routing problem with time windows pada distribusi minuman soda XYZ', *DORO: Repository Jurnal Mahasiswa PTIIK Universitas Brawijaya*, vol. 4, no. 1.
18. Haupt, RL & Haupt, SE 2004, *Practical Genetic Algorithm*, 2 edn, John Wiley & Sons, Inc., Hoboken, New Jersey.
19. Herrera, F, Lozano, M & Verdegay, JL 1998, 'Tackling real-coded genetic algorithms: operators and tools for behavioural analysis', *Artificial Intelligence Review*, vol. 12, pp. 265–319.
20. Hota, PK, Chakrabarti, R & Chattopadhyay, PK 1999, 'Short-term hydrothermal scheduling through evolutionary programming technique', *Electric Power Systems Research*, vol. 52, no. 2, 11/1/, pp. 189-196.
21. Hsu, C-M 2011, 'Applying genetic programming and ant colony optimisation to improve the geometric design of a reflector', *International Journal of Systems Science*, vol. 43, no. 5, 2012/05/01, pp. 972-986.
22. Ilmi, RR, Mahmudy, WF & Ratnawati, DE 2015, 'Optimasi penjadwalan perawat menggunakan algoritma genetika', *DORO: Repository Jurnal Mahasiswa PTIIK Universitas Brawijaya*, vol. 5, no. 13.
23. Im, S-M & Lee, J-J 2008, 'Adaptive crossover, mutation and selection using fuzzy system for genetic algorithms', *Artificial Life and Robotics*, vol. 13, no. 1, 2008/12/01, pp. 129-133.
24. Irwansyah, C, Pinandito, A & Mahmudy, WF 2014, 'Pencarian rute angkutan umum menggunakan algoritma ant colony optimization', *DORO: Repository Jurnal Mahasiswa PTIIK Universitas Brawijaya*, vol. 3, no. 10.
25. Kacem, I, Hammadi, S & Borne, P 2002, 'Approach by localization and multiobjective evolutionary optimization for flexible job-shop scheduling problems', *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, vol. 32, no. 1, pp. 1-13.
26. Khosraviani, B, Levitt, RE & Koza, JR 2004, Keijzer, M (ed), 'Organization design optimization using genetic programming', *Late-Breaking Papers at the 2004 Genetic and Evolutionary Computation Conference*, Seattle, WA, International Society of Genetic and Evolutionary Computation.
27. Koza, JR, Jones, LW, Keane, MA, Streeter, MJ & Al-Sakran, SH 2004, 'Toward automated design of industrial-strength analog circuits by means of genetic programming', Kluwer Academic Publishers, Boston, pp. 121–142.
28. Kusuma, JI, Mahmudy, WF & Indriati 2015, 'Optimasi komposisi pakan sapi potong menggunakan algoritma genetika', *DORO: Repository Jurnal Mahasiswa PTIIK Universitas Brawijaya*, vol. 5, no. 15.

29. Liliana, DY & Mahmudy, WF 2006, *Penerapan Algoritma Genetika pada Otomatisasi Penjadwalan Kuliah*, FMIPA Universitas Brawijaya, Malang.
30. Liqun, G, Feng, L, Yanfeng, G & Da, F 2010, 'A fuzzy adaptive Genetic Algorithms for global optimization problems', *Control and Decision Conference (CCDC), 2010 Chinese*, 26-28 May 2010, pp. 914-919.
31. Louchet, J 2000, 'Stereo analysis using individual evolution strategy', *Proceedings. 15th International Conference on Pattern Recognition*, pp. 908 – 911.
32. Lozano, M & Herrera, F 2003, 'Fuzzy adaptive genetic algorithms: design, taxonomy, and future directions', *Soft Computing*, vol. 7, pp. 545–562.
33. Lozano, M, Herrera, F, Krasnogor, N & Molina, D 2004, 'Real-coded memetic algorithms with crossover hill-climbing', *Evolutionary Computation*, vol. 12, no. 3, pp. 273-302.
34. Mahmudy, WF 2006, 'Penerapan algoritma genetika pada optimasi model penugasan', *Natural*, vol. 10, no. 3, pp. 197-207.
35. Mahmudy, WF 2007, 'Penyelesaian masalah transportasi menggunakan algoritma genetika dengan individu tunggal', *Kursor*, vol. 3, no. 1, pp. 30-36.
36. Mahmudy, WF 2008a, 'Optimasi multi travelling salesman problem (M-TSP) menggunakan algoritma genetika', *Seminar Nasional Basic Science V*, FMIPA, Universitas Brawijaya, Malang, 16 February.
37. Mahmudy, WF 2008b, 'Optimasi fungsi tanpa kendala menggunakan algoritma genetika dengan kromosom biner dan perbaikan kromosom hill-climbing', *Kursor*, vol. 4, no. 1, pp. 23-29.
38. Mahmudy, WF 2009, 'Optimasi fungsi tak berkendala menggunakan algoritma genetika terdistribusi dengan pengkodean real', *Seminar Nasional Basic Science VI* FMIPA, Universitas Brawijaya, Malang, 21 February.
39. Mahmudy, WF & Rahman, MA 2011, 'Optimasi fungsi multi-obyektif berkendala menggunakan algoritma genetika adaptif dengan pengkodean real', *Kursor*, vol. 6, no. 1, pp. 19-26.
40. Mahmudy, WF, Marian, RM & Luong, LHS 2012a, 'Solving part type selection and loading problem in flexible manufacturing system using real coded genetic algorithms – Part I: modeling', *International Conference on Control, Automation and Robotics*, Singapore, 12-14 September, World Academy of Science, Engineering and Technology, pp. 699-705.
41. Mahmudy, WF, Marian, RM & Luong, LHS 2012b, 'Solving part type selection and loading problem in flexible manufacturing system using real coded genetic algorithms – Part II: optimization', *International Conference on Control, Automation and Robotics*, Singapore, 12-14 September, World Academy of Science, Engineering and Technology, pp. 706-710.
42. Mahmudy, WF, Marian, RM & Luong, LHS 2013a, 'Real coded genetic algorithms for solving flexible job-shop scheduling problem – Part I: modeling', *Advanced Materials Research*, vol. 701, pp. 359-363.
43. Mahmudy, WF, Marian, RM & Luong, LHS 2013b, 'Real coded genetic algorithms for solving flexible job-shop scheduling problem – Part II: optimization', *Advanced Materials Research*, vol. 701, pp. 364-369.

44. Mahmudy, WF, Marian, RM & Luong, LHS 2013c, 'Hybrid genetic algorithms for multi-period part type selection and machine loading problems in flexible manufacturing system', *IEEE International Conference on Computational Intelligence and Cybernetics*, Yogyakarta, Indonesia, 3-4 December, pp. 126-130.
45. Mahmudy, WF, Marian, RM & Luong, LHS 2013d, 'Optimization of part type selection and loading problem with alternative production plans in flexible manufacturing system using hybrid genetic algorithms – Part 1: modelling and representation', *5th International Conference on Knowledge and Smart Technology (KST)*, Chonburi, Thailand, 31 Jan - 1 Feb, pp. 75-80.
46. Mahmudy, WF, Marian, RM & Luong, LHS 2013e, 'Optimization of part type selection and loading problem with alternative production plans in flexible manufacturing system using hybrid genetic algorithms – Part 2: genetic operators & results', *5th International Conference on Knowledge and Smart Technology (KST)*, Chonburi, Thailand, 31 Jan - 1 Feb, pp. 81-85.
47. Mahmudy, WF, Marian, RM & Luong, LHS 2013f, 'Solving integrated part type selection, machine loading and scheduling problems in FMS using hybrid genetic algorithms', *Submitted to: International Journal of Computer Integrated Manufacturing*.
48. Mahmudy, WF 2014a, 'Optimisation of Integrated Multi-Period Production Planning and Scheduling Problems in Flexible Manufacturing Systems (FMS) Using Hybrid Genetic Algorithms ', School of Engineering, University of South Australia.
49. Mahmudy, WF 2014b, 'Improved simulated annealing for optimization of vehicle routing problem with time windows (VRPTW)', *Kursor*, vol. 7, no. 3, pp. 109-116.
50. Mahmudy, WF 2014c, 'Optimasi part type selection and machine loading problems pada FMS menggunakan metode particle swarm optimization (Optimization of part type selection and machine loading problems in FMS using particle swarm optimization)', *Konferensi Nasional Sistem Informasi (KNSI)* STMIK Dipanegara, Makassar, 27 Februari - 1 Maret, pp. 1718-1723.
51. Mahmudy, WF, Marian, RM & Luong, LHS 2014, 'Hybrid genetic algorithms for part type selection and machine loading problems with alternative production plans in flexible manufacturing system', *ECTI Transactions on Computer and Information Technology (ECTI-CIT)*, vol. 8, no. 1, pp. 80-93.
52. Mahmudy, WF 2015a, 'Optimization of part type selection and machine loading problems in flexible manufacturing system using variable neighborhood search', *IAENG International Journal of Computer Science*, vol. 42, no. 3, pp. 254-264.
53. Mahmudy, WF 2015b, 'Improved particle swarm optimization untuk menyelesaikan permasalahan part type selection dan machine loading pada flexible manufacturing system (FMS) (Improved particle swarm optimization for solving part type selection and machine loading problems in flexible manufacturing system)', *Konferensi Nasional Sistem Informasi*, Universitas Klabat, Airmadidi, Minahasa Utara, Sulawesi Utara, 26-28 Februari, pp. 1003-1008.

54. Marian, RM 2003, 'Optimisation of assembly sequences using genetic algorithms', School of Advanced Manufacturing and Mechanical Engineering, University of South Australia.
55. Mawaddah, NK & Mahmudy, WF 2006, 'Optimasi penjadwalan ujian menggunakan algoritma genetika', *Kursor,* vol. 2, no. 2, pp. 1-8.
56. Michalewicz, Z 1996, *Genetic Algorithms + Data Structures = Evolution Programs*, Springer-Verlag, Heidelberg.
57. Milah, H & Mahmudy, WF 2015, 'Implementasi algoritma evolution strategies untuk optimasi komposisi pakan ternak sapi potong', *DORO: Repository Jurnal Mahasiswa PTIIK Universitas Brawijaya*, vol. 5, no. 11.
58. Muhlenbein, H & Schlierkamp-Voosen, D 1993, 'Predictive models for the breeder genetic algorithm; continuous parameter optimization', *Evolutionary Computation*, vol. 1, pp. 25–49.
59. Munawaroh, F & Mahmudy, WF 2015a, 'Penerapan algoritma evolution strategies untuk meminimumkan biaya distribusi barang', *DORO: Repository Jurnal Mahasiswa PTIIK Universitas Brawijaya*, vol. 5, no. 11.
60. Munawaroh, F & Mahmudy, WF 2015b, 'Optimasi distribusi pupuk menggunakan evolution strategies', *Journal of Environmental Engineering & Sustainable Technology*, vol. 2, no. 2, pp. 87-94.
61. Nissen, V & Günther, M 2009, 'Staff Scheduling with Particle Swarm Optimisation and Evolution Strategies', in Cotta, C & Cowling, P (eds), *Evolutionary Computation in Combinatorial Optimization*, vol. 5482, Springer Berlin Heidelberg, pp. 228-239.
62. Nurhumam, SD & Mahmudy, WF 2008, 'Optimasi multi travelling salesman problem (M-TSP) pada mobil patroli polisi dengan algoritma heuristic assignment Fisher-Jaikumar dan algoritma A*', *Kursor*, vol. 4, no. 1, pp. 15-22.
63. Ostertag, M, Nock, E & Kiencke, U 1995, 'Optimization of airbag release algorithms using evolutionary strategies', *Proceedings of the 4th IEEE Conference on Control Applications*, pp. 275 –280.
64. Panharesi, YG & Mahmudy, WF 2015, 'Optimasi distribusi barang dengan algoritma genetika', *DORO: Repository Jurnal Mahasiswa PTIIK Universitas Brawijaya*, vol. 5, no. 11.
65. Permatasari, AI & Mahmudy, WF 2015, 'Pemodelan regresi linear dalam konsumsi Kwh listrik di Kota Batu menggunakan algoritma genetika ', *DORO: Repository Jurnal Mahasiswa PTIIK Universitas Brawijaya*, vol. 5, no. 14.
66. Pezzella, F, Morganti, G & Ciaschetti, G 2008, 'A genetic algorithm for the flexible job-shop scheduling problem', *Computers & Operations Research*, vol. 35, no. 10, pp. 3202-3212.
67. Pitaloka, DA, Mahmudy, WF & Sutrisno 2014, 'Penyelesaian vehicle routing problem with time windows (VRPTW) menggunakan algoritma genetika hybrid', *Journal of Environmental Engineering & Sustainable Technology*, vol. 1, no. 2, pp. 103-109.
68. Pratiwi, MI, Mahmudy, WF & Dewi, C 2014, 'Implementasi algoritma genetika pada optimasi biaya pemenuhan kebutuhan gizi', *DORO: Repository Jurnal Mahasiswa PTIIK Universitas Brawijaya*, vol. 4, no. 6.

69. Putri, AMDA, Mahmudy, WF & Cholissodin, I 2015, 'Optimasi model fuzzy AHP dengan menggunakan algoritma evolution strategies (studi kasus: pemilihan calon penerima beasiswa PTIIK Universitas Brawijaya)', *DORO: Repository Jurnal Mahasiswa PTIIK Universitas Brawijaya*, vol. 5, no. 15.
70. Putri, FB, Mahmudy, WF & Ratnawati, DE 2015, 'Penerapan algoritma genetika untuk vehicle routing problem with time windows (VRPTW) pada kasus optimasi distribusi beras bersubsidi', *DORO: Repository Jurnal Mahasiswa PTIIK Universitas Brawijaya*, vol. 5, no. 1.
71. Qi, JG, Burns, GR & Harrison, DK 2000, 'The application of parallel multipopulation genetic algorithms to dynamic job-shop scheduling', *The International Journal of Advanced Manufacturing Technology*, vol. 16, no. 8, pp. 609-615.
72. Rahmi, A, Mahmudy, WF & Setiawan, BD 2015, 'Prediksi harga saham berdasarkan data historis menggunakan model regresi yang dibangun dengan algoritma genetika', *DORO: Repository Jurnal Mahasiswa PTIIK Universitas Brawijaya*, vol. 5, no. 12.
73. Ramuna, MDT & Mahmudy, WF 2015, 'Optimasi persediaan barang dalam poduksi jilbab mnggunakan agoritma genetika ', *DORO: Repository Jurnal Mahasiswa PTIIK Universitas Brawijaya*, vol. 5, no. 14.
74. Rianawati, A & Mahmudy, WF 2015, 'Implementasi algoritma genetika untuk optimasi komposisi makanan bagi penderita diabetes mellitus', *DORO: Repository Jurnal Mahasiswa PTIIK Universitas Brawijaya*, vol. 5, no. 14.
75. Rothlauf, F 2006, *Representations for Genetic and Evolutionary Algorithms*, Springer, Berlin Heidelberg.
76. Samaher & Mahmudy, WF 2015, 'Penerapan algoritma genetika untuk memaksimalkan laba produksi jilbab', *Journal of Environmental Engineering & Sustainable Technology*, vol. 2, no. 1, pp. 6-11.
77. Samuel, GG & Rajan, CCA 2013, 'Hybrid particle swarm optimization: Evolutionary programming approach for solving generation maintenance scheduling problem', *Scientific Research and Essays*, vol. 8, no. 35, pp. 1701-1713.
78. Saputri, MW, Mahmudy, WF & Ratnawati, DE 2015, 'Optimasi vehicle routing problem with time window (VRPTW) menggunakan algoritma genetika pada distribusi barang', *DORO: Repository Jurnal Mahasiswa PTIIK Universitas Brawijaya*, vol. 5, no. 12.
79. Saputro, HA, Mahmudy, WF & Dewi, C 2015, 'Implementasi algoritma genetika untuk optimasi penggunaan lahan pertanian', *DORO: Repository Jurnal Mahasiswa PTIIK Universitas Brawijaya*, vol. 5, no. 12.
80. Sari, AP, Mahmudy, WF & Dewi, C 2014, 'Optimasi asupan gizi pada ibu hamil dengan menggunakan algoritma genetika', *DORO: Repository Jurnal Mahasiswa PTIIK Universitas Brawijaya*, vol. 4, no. 5.
81. Sari, DDP, Mahmudy, WF & Ratnawati, DE 2015, 'Optimasi penjadwalan mata pelajaran menggunakan algoritma genetika (studi kasus : SMPN 1 Gondang Mojokerto) ', *DORO: Repository Jurnal Mahasiswa PTIIK Universitas Brawijaya*, vol. 5, no. 13.

82. Sari, RN & Mahmudy, WF 2015, 'Penyelesaian multiple travelling salesperson problem (M-TSP) dengan algoritma genetika : studi kasus pendistribusian air mineral', *DORO: Repository Jurnal Mahasiswa PTIIK Universitas Brawijaya,* vol. 5, no. 14.
83. Schwefel, H-P 1995, *Evolution and Optimum Seeking*, Wiley, New York.
84. Sebt, MH & Alipouri, Y 2013, 'Solving Resource-Constrained Project Scheduling Problem with Evolutionary Programming', *Journal of the Operational Research Society,* vol. 64, no. 9, pp. 1327-1335.
85. Seputro, DH, Mahmudy, WF & Mursityo, YT 2015, 'Pengembangan sistem informasi pembacaan meter berbasis metode tabu search (Studi kasus PT. PLN persero distribusi Jawa Timur area Malang rayon Dinoyo)', *DORO: Repository Jurnal Mahasiswa PTIIK Universitas Brawijaya,* vol. 5, no. 16.
86. Smith, JE & Eiben, AE 2003, *Introduction to evolutionary computing*, Springer, London :.
87. Spector, L 2004, *Automatic Quantum Computer Programming: A Genetic Programming Approach*, Kluwer Academic Publishers, Boston.
88. Sulistiyorini, R & Mahmudy, WF 2015, 'Penerapan algoritma genetika untuk permasalahan optimasi distribusi barang dua tahap', *DORO: Repository Jurnal Mahasiswa PTIIK Universitas Brawijaya,* vol. 5, no. 12.
89. Taha, HA 2011, *Operations research: an introduction*, 9 edn, Prentice Hall, Upper Saddle River, N.J.
90. Vista, CB & Mahmudy, WF 2015, 'Penerapan algoritma evolution strategies untuk optimasi distribusi barang dua tahap', *DORO: Repository Jurnal Mahasiswa PTIIK Universitas Brawijaya,* vol. 5, no. 11.
91. Wahid, N & Mahmudy, WF 2015, 'Optimasi komposisi makanan untuk penderita kolesterol menggunakan algoritma genetika', *DORO: Repository Jurnal Mahasiswa PTIIK Universitas Brawijaya,* vol. 5, no. 15.
92. Widodo, AW & Mahmudy, WF 2010, 'Penerapan algoritma genetika pada sistem rekomendasi wisata kuliner', *Kursor,* vol. 5, no. 4, pp. 205-211.
93. Yazdani, M, Amiri, M & Zandieh, M 2010, 'Flexible job-shop scheduling with parallel variable neighborhood search algorithm', *Expert Systems with Applications,* vol. 37, no. 1, pp. 678-687.
94. Yogeswaran, M, Ponnambalam, SG & Tiwari, MK 2009, 'An efficient hybrid evolutionary heuristic using genetic algorithm and simulated annealing algorithm to solve machine loading problem in FMS', *International Journal of Production Research,* vol. 47, no. 19, pp. 5421-5448.
95. Zhang, G, Shao, X, Li, P & Gao, L 2009, 'An effective hybrid particle swarm optimization algorithm for multi-objective flexible job-shop scheduling problem', *Computers & Industrial Engineering*, vol. 56, no. 4, pp. 1309-1318.

Indeks

algoritma evolusi, 2
algoritma genetika, 5, 10
analog circuit, 87
ant colony optimisation, 87
assembly stage, 51
binary tournament, 14
chromosome, 10
constrains, 55
contraint, 48, 56
cycle crossover, 49
decision tree, 93
decoding, 9
deterministic sampling, 38
distributed genetic algorithms, 71
elitism, 14, 39
elitism selection, 35, 44
encoding, 9
ergodic, 9
evolution strategies, 75
Evolution Strategies, 5
evolutionary algorithms, 2
Evolutionary Programming, 5, 92
extended intermediate crossover, 30
fitness, 4
flow-shop scheduling problem, 50
FSP, 50
Gantt-Chart, 51
gen, 9
genetic algorithms, 7
Genetic Algorithms, 5
Genetic Programming, 5
heuristic crossover, 49
heuristik, 2
hill-climbing, 2, 69
hybrid GAs, 9
Hybrid Genetic Algorithms, 69
individu, 10
insertion mutation, 50, 54
job-based representation, 53, 54
job-shop scheduling problem, 53
JSP, 53
kendala, 48, 56
koloni semut, 2
komputasi evolusi, 2
lambda, 76
linear programming, 55
local search, 69
makespan, 51
MAs, 69
memetic algorithms, vii, 69
meta-heuristic, 2
migrasi, 71
mixed sampling, 38
natural selection, 4
offspring, 4
one-cut-point crossover, 13, 18
optimasi, 2
order crossover, 49
order-based crossover, 49
parallel genetic algorithms, 71
parent, 4
partial-mapped crossover, 49
particle swarm optimization, 75, 92
population based, 5
position-based crossover, 49
precedence, 50
quantum computing circuits, 87
random mutation, 30
RCGA, 70
real-coded GAs, 70
real-coded genetic algorithms, 29
reciprocal exchange mutation, 50, 54
rekombinasi, 76
repairing, 57
replacement selection, 35, 44
roulette wheel, 14, 15, 20
search space, 8
seleksi alam, 4

self-adaptation, 75
simulated annealing, 2, 87
stochastic, 5
stochastic operators, 8
stochastic sampling, 38
sub-populasi, 8
tabu search, 2
termination condition, 25
tournament selection, 31
Transportation Problem, 54
Travelling Salesperson Problem, 47
two-stage assembly flowshop, 51
variable neighbourhoods search, 70
VNS, 70