BAB 3 PERANCANGAN

Pada bab ini dibahas mengenai definisi variabel, *domain*, dan *constraint* penjadwalan kuliah sebagai CSP pada subbab 3.1 dan 3.2, representasi *chromosome* pada subbab 3.3, rancangan *fitness function* pada subbab 3.4, *crossover* dan *mutation* pada subbab 3.5, dan rancangan eksperimen pada subbab 3.6.

3.1 Definisi Variabel dan Domain Penjadwalan Kuliah Sebagai CSP

Komponen penyusun jadwal yaitu mata kuliah (MK), dosen, ruang, dan slot. Setiap MK memiliki informasi:

- nama MK,
- bobot sks (sistem kredit semester),
- tingkat MK tersebut ditawarkan,
- wajib atau tidaknya MK tersebut,
- hubungan prasyarat dengan MK lain,
- jumlah pengikut MK tersebut,
- tingkat kefavoritan MK tersebut,
- tingkat kecenderungan MK tersebut disodok,
- tingkat kecenderungan MK tersebut diulang.

Setiap dosen memiliki informasi:

- nama dosen,
- daftar MK yang dikuasai (ekspertise) dosen tersebut,
- kapan saja waktu dosen tersebut lowong,
- maksimum sks yang bisa dibebankan terhadap dosen tersebut.

Setiap ruang memiliki informasi:

- nama ruangan,
- kapasitas yang dapat ditampung ruangan tersebut.

Setiap slot memiliki informasi:

- hari kapan slot tersebut berlangsung,
- jam kapan slot tersebut berlangsung,
- kategori waktu sebelum makan siang atau sesudah.

Pada sistem sks yang berlaku di Fasilkom, 1 sks memiliki jatah 1 slot, jadi MK dengan 3 sks memiliki jatah waktu perkuliahan 3 slot. Sehingga, berbeda dengan definisi penjadwalan kuliah secara umum pada subbab 2.2, pada kasus Fasilkom UI penjadwalan kuliah dapat dapat dimodelkan sebagai CSP dengan variabel X_1, X_2, …, X_n di mana n = total jumlah sks MK yang ditawarkan pada suatu semester dan $X_i = \{ (m_i, d_i, r_i, s_i) \mid 1 \leq i \leq n, i \in$ bilangan bulat, $m_i \in$ MK, $d_i \in$ dosen, $r_i \in$ ruangan, $s_i \in$ slot$\}$. Dengan pengertian 1 sks dari MK m_i diajar oleh dosen d_i di ruangan r_i pada slot s_i. Dengan kata lain, satu *tuple* (m_i, d_i, r_i, s_i) menyatakan kuliah pada satu slot tertentu. Agar lebih mudah dipahami, selanjutnya diberikan contoh kasus yang akan digunakan sepanjang bab 3 ini.

Misalnya ada sebuah fakultas bernama Mini Fasilkom menawarkan 5 MK dalam satu semester. Dosen yang mengajar di fakultas ini jumlahnya ada 4 orang. Ruangan yang tersedia untuk kegiatan perkuliahan ada 3 ruangan. Waktu perkuliahan hanya di hari Senin dan Selasa yang totalnya ada 16 slot. Informasi yang lebih detil mengenai MK ada pada Tabel 3.1, dosen pada Tabel 3.2, ruangan pada Tabel 3.3, dan waktu perkuliahan (slot) pada Tabel 3.4.

Tabel 3.1 Informasi Mata Kuliah Mini Fasilkom

Kode	Nama	S	P	T	W	F	N	U
0	IKI10820 Dasar-Dasar Pemrograman A	4	60	1	Ya	1	0	1
1	IKI10820 Dasar-Dasar Pemrograman B	4	60	1	Ya	1	0	1
2	IKI10201 Pengantar Sistem Digital	4	120	1	Ya	1	0	1
3	IKI20100 Struktur Data & Algoritma	4	120	2	Ya	1	0	1
4	IKI20200 Organisasi Sistem Komputer	3	60	2	Ya	1	1	0

Keterangan Tabel 3.1:

S = bobot sks

P = jumlah pengikut

T = tingkat

W = wajib/tidak wajib

F = tingkat kefavoritan

N = tingkat kecenderungan MK disodok

U = tingkat kecenderungan MK diulang

Tabel 3.2 Informasi Dosen Mini Fasilkom

Kode	Nama	Ekspertise	Lowong	Maks Sks
0	Halsen	[0, 1]	[0, 1, 2, 3, 8, 9, 10, 11]	8
1	Kivin	[2]	[4, 5, 12, 13]	4
2	Rus	[3]	[0, 1, 8, 9]	4
3	Yuba	[4]	[4, 5, 12]	3

Tabel 3.3 Informasi Ruangan Mini Fasilkom

Kode	Nama	Kapasitas
0	R 2102	130
1	R 2301	65
2	R 2302	75

Tabel 3.4 Informasi Slot Mini Fasilkom

Kode	Hari & Jam	Sebelum makan siang?
0	Senin(08.00-09.00)	Ya
1	Senin(09.00-10.00)	Ya
2	Senin(10.00-11.00)	Ya
3	Senin(11.00-12.00)	Ya
4	Senin(13.00-14.00)	Tidak
5	Senin(14.00-15.00)	Tidak
6	Senin(15.00-16.00)	Tidak
7	Senin(16.00-17.00)	Tidak
8	Selasa(08.00-09.00)	Ya
9	Selasa(09.00-10.00)	Ya
10	Selasa(10.00-11.00)	Ya
11	Selasa(11.00-12.00)	Ya
12	Selasa(13.00-14.00)	Tidak
13	Selasa(14.00-15.00)	Tidak
14	Selasa(15.00-16.00)	Tidak
15	Selasa(16.00-17.00)	Tidak

Informasi prasyarat MK pada Mini Fasilkom yaitu MK 0 dan 1 merupakan prasyarat dari MK 3, dan MK 2 merupakan prasyarat dari MK 4. Kolom ekspertise pada Tabel 3.2 berisi *list* MK yang merupakan ekspertise dosen yang bersangkutan. Misalnya pak Halsen memiliki ekspertise di MK 0 (IKI10820 Dasar-Dasar Pemrograman A) dan MK 1 (IKI10820 Dasar-Dasar Pemrograman B). Begitu juga dengan kolom lowong pada Tabel 3.2 berisi *list* slot kapan waktu

dosen yang bersangkutan lowong. Misalnya pak Kivin memiliki waktu lowong di slot 4 (Senin(13.00-14.00)), 5 (Senin(14.00-15.00)), 12 (Selasa(13.00-14.00)), dan 13 (Selasa(14.00-15.00)).

(0, 0, 1, 2),
(0, 0, 1, 3),
(0, 0, 1, 10),
(0, 0, 1, 11),
(1, 0, 1, 0),
(1, 0, 1, 1),
(1, 0, 2, 8),
(1, 0, 2, 9),
(2, 1, 0, 4),
(2, 1, 0, 5),
(2, 1, 0, 12),
(2, 1, 0, 13),
(3, 2, 0, 0),
(3, 2, 0, 1),
(3, 2, 0, 8),
(3, 2, 0, 9),
(4, 3, 2, 12),
(4, 3, 2, 4),
(4, 3, 2, 5)

Gambar 3.1 Contoh *Assignment* Lengkap Pada Kasus Mini Fasilkom

Variabel CSP untuk kasus ini adalah $X_1, X_2, \ldots, X_n$ di mana $n = 19$ karena total jumlah sks dari kelima MK yang ditawarkan adalah 4 + 4 + 4 + 4 + 3 = 19 dan $X_i = \{ (m_i, d_i, r_i, s_i) \mid 0 < i \leq 19, m_i \in \text{MK } (\{0, 1, 2, 3, 4\}), d_i \in \text{dosen } (\{0, 1, 2, 3\}), r_i \in \text{ruangan } (\{0, 1, 2\}), s_i \in \text{slot } (\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15\})\}$. Variabel CSP untuk kasus Mini Fasilkom dapat dituliskan sebagai berikut.

(m_1, d_1, r_1, s_1),
(m_2, d_2, r_2, s_2),
.
.
$(m_{19}, d_{19}, r_{19}, s_{19})$

Suatu *state* dapat diperoleh dari *assignment* lengkap terhadap variabel CSP. Contoh *assignment* lengkap pada kasus Mini Fasilkom dapat dilihat pada Gambar 3.1. Arti dari *tuple* (0, 0, 1, 2) adalah MK 0 (IKI10820 Dasar-Dasar Pemrograman A) diajar oleh dosen 0 (pak Halsen) di ruang 1 (R 2301) pada slot 2 (Senin(10.00-11.00)).

3.2 Definisi Constraint Penjadwalan Kuliah Sebagai CSP

Solusi dari masalah penjadwalan adalah terbentuknya jadwal yang memenuhi semua *constraint* yang ada, atau dengan kata lain, konfigurasi MK, dosen, ruang dan slot sedemikian sehingga memenuhi semua *hard constraint* dan sebanyak mungkin *soft constraint* yang ada. *Hard constraint* (HC) berarti *constraint* yang sama sekali tidak boleh dilanggar karena jika dilanggar akan menyebabkan jadwal tidak valid, sementara *soft constraint* (SC) masih boleh dilanggar tetapi sebaiknya dipenuhi. Berikut ini definisi variabel secara lebih matematis dan penjelasan masing-masing *constraint*.

$M = \{MataKuliah_1, MataKuliah_2, ..., MataKuliah_p\}$

p = jumlah MK yang ditawarkan pada suatu semester

$D = \{Dosen_1, Dosen_2, ..., Dosen_q\}$

q = jumlah dosen

$R = \{Ruang_1, Ruang_2, ..., Ruang_t\}$

t = jumlah ruang

$S = \{Slot_1, Slot_2, ..., Slot_u\}$

u = jumlah slot

Variabel CSP: $X_1, X_2, \ldots, X_n$

n = total jumlah sks MK yang ditawarkan pada suatu semester

I = bilangan bulat

$1 \le i,j \le n \; ; \; i,j \in I$

$X_i = \{ (m_i, d_i, r_i, s_i) \mid m_i \in M, d_i \in D, r_i \in R, s_i \in S \}$

$waktu(x) = \{s \mid s \in S \wedge x \text{ diajarkan pada slot } s \text{ berdasarkan } X\}, x \in M$

$$potong(x,y)\begin{cases} true \;\; jika \;\; y \subseteq x \\ false \;\; jika \;\; x \subset y \end{cases} x, y \in S$$

Hard constraint (HC):

1. Jadwal semua MK sesuai dengan jumlah sksnya. *Constraint* ini selanjutnya disebut HC1. Definisi matematis *constraint* ini:

   $\forall a \ sks(a) = \sum_{k=1}^{n} mksama(m_k, a), a \in M$

   $sks(x) \in \{1,2,3,4\}, x \in M$ (didapat dari asumsi jenis bobot sks yang mungkin adalah 1 sampai dengan 4)

   $$mksama(x,y)\begin{cases}1 & jika \ x = y \\ 0 & jika \ x \neq y\end{cases} \quad x, y \in M$$

2. MK wajib di tingkat yang sama tidak bentrok. *Constraint* ini selanjutnya disebut HC2. Definisi matematis *constraint* ini:

   $\forall i, j \ i \neq j \wedge tingkat(m_i) = tingkat(m_j) \wedge wajib(m_i) \wedge wajib(m_j) \rightarrow s_i \neq s_j$

   $tingkat(x) \in \{1,2,3,4\}, x \in M$ (didapat dari asumsi jenis tingkat)

   $wajib(x) \in \{true, false\}, x \in M$

3. Ruang dan slot yang sama tidak diisi lebih dari satu MK. *Constraint* ini selanjutnya disebut HC3. Definisi matematis *constraint* ini:

   $\forall i, j \ i \neq j \wedge (r_i \neq r_j \vee s_i \neq s_j)$

4. Dosen tidak mengajar lebih dari satu MK pada slot yang sama. *Constraint* ini selanjutnya disebut HC4. Definisi matematis *constraint* ini:

   $\forall i, j \ i \neq j \wedge (d_i \neq d_j \vee s_i \neq s_j)$

5. Jumlah pengikut suatu MK lebih kecil atau sama dengan kapasitas ruangannya. *Constraint* ini selanjutnya disebut HC5. Definisi matematis *constraint* ini:

   $\forall i \ \ pengikut(m_i) \leq kapasitas(r_i)$

   $pengikut(x) \in I, x \in M$ (karena tidak ada asumsi maksimum/minimum pengikut kuliah)

   $kapasitas(x) \in I, x \in R$ (karena tidak ada asumsi maksimum/minimum kapasitas ruangan)

Soft constraint (SC):

1. Dosen mengajar tidak melebihi maksimum beban sksnya. *Constraint* ini selanjutnya disebut SC1. Definisi matematis *constraint* ini:

   $\forall b\ maksbeban(b) = \sum_{k=1}^{n} dosensama(d_k, b), b \in D$

   $maksbeban(x) \in I, x \in D$ (karena tidak ada asumsi maksimum/minimum beban dosen)

   $$dosensama(x, y)\begin{cases} 1 & jika\ x = y \\ 0 & jika\ x \neq y \end{cases} x, y \in D$$

2. Dosen hanya mengajar kuliah yang dia ekspertise. *Constraint* ini selanjutnya disebut SC2. Definisi matematis *constraint* ini:

   $\forall i\ m_i \in ekspertise(d_i)$

   $ekspertise(x) \subseteq M, x \in D$

3. Dosen hanya mengajar pada waktu lowongnya. *Constraint* ini selanjutnya disebut SC3. Definisi matematis *constraint* ini:

   $\forall i\ s_i \in lowong(d_i)$

   $lowong(x) \subseteq S, x \in D$

4. Kuliah tidak terpotong makan siang. *Constraint* ini selanjutnya disebut SC4. Definisi matematis *constraint* ini:

   $\forall a, b \neg potong(waktu(a), b), a \in M, b \in V$

   v = jumlah jeda makan siang

   $V = \{V_1, V_2, ..., V_v\}$

   $V_c = \{x_c, y_c\}, x_c \neq y_c, x_c \in S, y_c \in S, 1 \leq c \leq v, c \in I$

5. Kuliah tidak terpotong ke hari berikutnya. *Constraint* ini selanjutnya disebut SC5. Definisi matematis *constraint* ini:

   $\forall a, b \neg potong(waktu(a), b), a \in M, b \in W$

   w = jumlah jeda hari

   $W = \{W_1, W_2, ..., W_w\}$

   $W_c = \{x_c, y_c\}, x_c \neq y_c, x_c \in S, y_c \in S, 1 \leq c \leq w, c \in I$

6. Kuliah bentrok sebisa mungkin tidak mengganggu mahasiswa mengambil kuliah yang dia inginkan. *Constraint* ini selanjutnya disebut SC6. Definisi matematis *constraint* ini:

$$\forall a,b\ bentrok(waktu(a),waktu(b)) \rightarrow hubprasyarat(a,b), a \in M, b \in M$$

$$bentrok(A,B)\begin{cases} true\ \ jika\ \ A \cap B \neq \phi \\ false\ \ jika\ \ A \cap B = \phi \end{cases} A,B \subseteq S$$

z = jumlah hubungan prasyarat

$Z = \{Z_1, Z_2, ...,Z_z\}$

$Z_c = \{x_c,y_c\}, x_c \neq y_c, x_c \in M, y_c \in M, 1 \leq c \leq z, c \in I$

$$hubprasyarat(x,y)\begin{cases} true\ \ jika\ \ \{x,y\} \in Z \\ false\ \ jika\ \ \{x,y\} \notin Z \end{cases} x,y \in M$$

Constraint dan asumsi khusus Fasilkom UI:

- Jadwal MPKT, olahraga, atau seni tidak bisa diubah. (tidak masuk *scope*).
- Rabu jam 13.00 s.d 14.00 tidak boleh dipakai untuk kuliah karena ada seminar Reboan.
- Jumat jam 13.00 s.d. 15.00 tidak ada kuliah oleh dosen dalam Fasilkom UI, karena pada jam tersebut ada pertemuan akademis.
- Setiap hari kuliah (Senin hingga Jumat) jam istirahat adalah jam 12.00 s.d. 13.00.
- Dosen menyukai ruangan yang sama untuk semua kuliah yang diajarkannya. (tidak masuk *scope*).
- Satu MK dapat diajar oleh satu (atau lebih) orang dosen.
- Satu MK dapat dipecah menjadi 2 atau 3 kelas. Misalnya MK Dasar-Dasar Pemrograman (DDP) dapat dipecah menjadi DDP A dan DDP B.
- MK mendapat jatah slot sesuai dengan bobot sksnya. MK dengan bobot 3 sks mendapat jatah 3 slot.
- MK dengan bobot sks > 2 dipecah menjadi 2 kali pertemuan. Jika MK memikiki bobot 3 sks, MK ini dipecah menjadi 2 kali pertemuan, 1 pertemuan berdurasi 2 slot dan 1 pertemuan lainnya berdurasi 1 slot.
- Jeda satu kuliah yang sama tidak terlalu pendek. (tidak masuk *scope*).
- Bobot sks MK yang ada adalah antara 1 sampai dengan 4 sks.
- Tingkat MK yang ada adalah antara tingkat 1 sampai dengan tingkat 4.

Jika dibandingkan dengan *constraint* yang digunakan Soraya (lihat subbab 2.1) ada beberapa *constraint* yang baru dan ada yang tidak dicakup tugas akhir ini. *Constraint* yang baru (tidak dicakup penelitian Soraya) antara lain:

- ruang dan slot yang sama tidak diisi lebih dari satu MK,
- dosen hanya mengajar pada waktu lowongnya,
- kuliah tidak terpotong makan siang,
- kuliah tidak terpotong ke hari berikutnya,
- kuliah bentrok sebisa mungkin tidak mengganggu mahasiswa mengambil kuliah yang dia inginkan.

Sementara, *constraint* yang tidak dicakup tugas akhir ini tetapi dicakup penelitian Soraya antara lain:

- dosen mengajar MK karena permintaan dosen atau karena ditunjuk oleh Fakultas melalui bagian akademis,
- permintaan khusus dari dosen, seperti ruang kelas yang diinginkan terpenuhi.

Penyelesaian penjadwalan kuliah pada tugas akhir ini hanya dilakukan dalam 1 tahap, berbeda dengan yang dilakukan oleh Soraya (lihat subbab 2.1) yang membagi penyelesaian dalam 2 tahap. Penyelesaian dalam 1 tahap memiliki *search space* yang lebih besar dari penyelesaian dalam 2 tahap karena masalah pemetaan MK dengan dosen dan masalah pemetaan MK dengan slot dilakukan bersamaan, padahal keduanya saling bergantung (masalah yang *dependent*). Penyelesaian dalam 1 tahap memiliki keuntungan dan kerugian dibandingkan dengan penyelesaian dalam 2 tahap. Keuntungannya antara lain:

- tidak menutup kemungkin menemukan solusi optimal yang disebabkan pembatasan *search space* ketika pemetaan MK dengan dosen sudah *commit* lebih dulu (seperti pada penyelesaian dalam 2 tahap),
- lebih mungkin mendapatkan solusi yang lebih baik atau minimal sama karena *search space* penyelesaian dalam 2 tahap tercakup dalam *search space* penyelesaian dalam 1 tahap.

Sementara, kerugiannya adalah:

- kecepatan pencarian solusi lebih lambat karena *search space* yang dimiliki lebih besar.

Pencarian solusi dari penjadwalan kuliah sebagai CSP dalam tugas akhir ini menggunakan *Genetic algorithm* (GA) (lihat subbab 2.3). Oleh karena itu, perlu adanya representasi *chromosome* yang menyatakan *state* pada masalah penjadwalan kuliah dan *fitness function* yang menilai suatu *chromosome* (individu) untuk mengarahkan generasi ke arah yang lebih baik. Representasi *chromosome* dijelaskan di subbab 3.3 dan rancangan *fitness function* dijelaskan di subbab 3.4.

3.3 Representasi Chromosome

Untuk menggunakan GA dalam masalah penjadwalan diperlukan representasi jadwal ke *chromosome*. Ada berbagai cara dalam merepresentasikan jadwal ke *chromosome*. Pada tugas akhir ini dibahas 4 representasi, yaitu representasi MDRS, DRS, DRSRSK, dan DRSRS. Representasi MDRS merupakan *fenotype* dan ketiga representasi lainnya merupakan *genotype*. Pembahasan mengenai *fenotype* dan *genotype* dapat dilihat pada subbab 2.3. Simbol M menandakan mata kuliah (MK), D menandakan dosen, R menandakan ruang, S menandakan Slot, dan K menandakan kode. Penjelasan lebih lanjut mengenai kode terdapat pada subbab 3.3.3 Representasi DRSRSK.

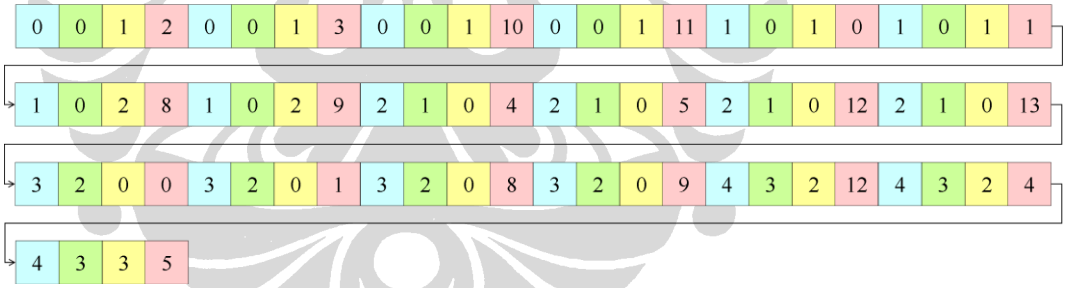
Lebih lanjut, masing-masing simbol memiliki *domain* masing-masing. *Domain* dari M adalah bilangan bulat dari 0 sampai dengan jumlah MK - 1. *Domain* dari D adalah bilangan bulat dari 0 sampai dengan jumlah dosen - 1. *Domain* dari R adalah bilangan bulat dari 0 sampai dengan jumlah ruangan - 1. *Domain* dari S adalah bilangan bulat dari 0 sampai dengan jumlah slot - 1. Sementara *domain* K adalah bilangan 0 atau 1. Selanjutnya akan dibahas lebih detil mengenai masing-masing representasi, serta kekurangan dan kelebihan masing-masing representasi.

3.3.1 Representasi MDRS

Pada representasi ini masing-masing *tuple* nilai $X_i = \{ (m_i, d_i, r_i, s_i) \mid 1 \leq i \leq n, i \in$ bilangan bulat, $m_i \in M, d_i \in D, r_i \in R, s_i \in S\}$ dinyatakan secara ekplisit sebagai 4 nilai integer yang menyatakan kode MK, kode dosen, kode ruang, dan kode slot. Sehingga, panjang *chromosome* untuk merepresentasikan jadwal dalam representasi ini adalah 4 x total sks (n). Jika ada 2 MK, masing-masing berbobot 3 sks maka total sks adalah 2 x 3 = 6 dan panjang *chromosome* adalah 4 x 6 = 24. Ilustrasi dari representasi MDRS dapat dilihat pada Gambar 3.2.

Gambar 3.2 Representasi MDRS

Jika representasi ini diterapkan pada contoh kasus Mini Fasilkom pada subbab 3.2, panjang *chromosome* yang terjadi adalah 4 x 19 = 76. Ilustrasi dari representasi MDRS untuk *assignment* lengkap yang sesuai dengan contoh pada kasus Mini Fasilkom dapat dilihat pada Gambar 3.3. Tanda panah menunjukkan sambungan dari *chromosome*.

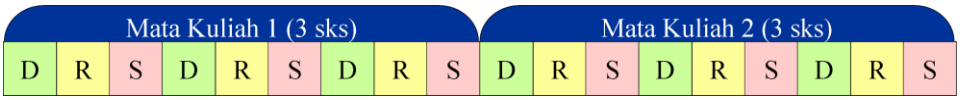


Gambar 3.3 Representasi MDRS pada Contoh Kasus Mini Fasilkom

3.3.2 Representasi DRS

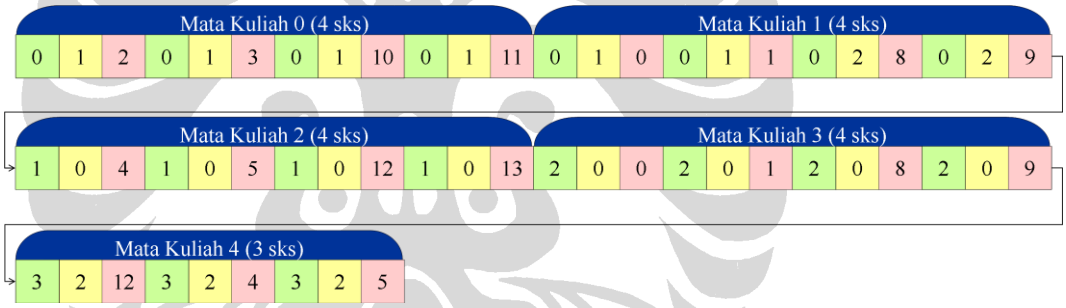
Representasi ini adalah modifikasi dari representasi MDRS. Pada representasi ini urutan dan porsi MK sudah ditentukan. Hal ini untuk menjamin HC1 (lihat subbab 3.2) terpenuhi. Panjang *chromosome* untuk merepresentasikan jadwal dalam representasi ini adalah 3 x total sks. Jika ada 2 MK, masing-masing berbobot 3 sks maka total sks adalah 2 x 3 = 6 dan panjang *chromosome* adalah 3 x 6 = 18. Cara menentukan MK mana yang berkorespondensi dengan suatu DRS yaitu dengan aturan MK pertama berkorespondensi dengan m DRS pertama, di mana m adalah

bobot sks MK pertama. Selanjutnya, MK kedua berkorespondensi dengan *n* DRS berikutnya setelah DRS ke-*m*, di mana *n* adalah bobot sks MK kedua. Ilustrasi dari representasi DRS dapat dilihat pada Gambar 3.4.



Gambar 3.4 Representasi DRS

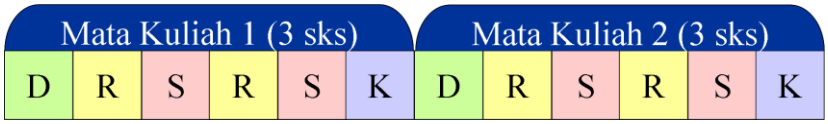
Jika representasi ini diterapkan pada contoh kasus Mini Fasilkom pada subbab 3.2, panjang *chromosome* yang terjadi adalah 3 x 19 = 57. Ilustrasi dari representasi DRS untuk *assignment* lengkap yang sesuai dengan contoh pada kasus Mini Fasilkom dapat dilihat pada Gambar 3.5. Tanda panah menunjukkan sambungan dari *chromosome*.



Gambar 3.5 Representasi DRS pada Contoh Kasus Mini Fasilkom

3.3.3 Representasi DRSRSK

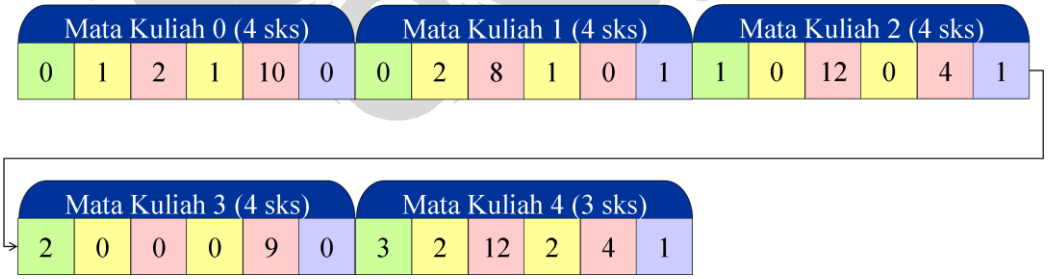
Representasi ini adalah modifikasi dari representasi DRS, sehingga tetap menjamin HC1 terpenuhi. Pada representasi ini pertemuan suatu MK dibagi menjadi 2 pertemuan dengan dosen, ruang dan slot yang tetap pada setiap pertemuannya. Panjang *chromosome* untuk merepresentasikan jadwal dalam representasi ini adalah 6 x jumlah MK. Jika ada 2 MK, masing-masing berbobot 3 sks maka panjang *chromosome* adalah 6 x 2 = 12. Ilustrasi dari representasi DRSRSK dapat dilihat pada Gambar 3.6.



Gambar 3.6 Representasi DRSRSK

Dalam representasi ini, satu MK diajar oleh satu dosen (D) dan pertemuan (RS) dibagi menjadi maksimal 2 kali, yaitu RS pertama dan RS kedua. Di sini, kode (K) berfungsi sebagai penanda bagaimana bobot sks didistribusikan ke RS. Untuk kasus MK dengan bobot 3 sks, jika nilai K = 0 maka 2 sks didistribusikan ke RS pertama dan 1 sks didistribusikan ke RS kedua. Sementara, jika nilai K = 1 maka 2 sks didistribusikan ke RS kedua dan 1 sks didistribusikan ke RS pertama. Pada kasus MK dengan bobot 1 atau 2 sks, jika nilai K = 0 maka pertemuan diadakan pada RS pertama, sebaliknya jika nilai K = 1 maka pertemuan diadakan pada RS kedua. Terakhir, pada kasus MK dengan bobot 4 sks, kedua pertemuan akan mendapat jatah masing-masing 2 sks tidak peduli berapapun nilai K.

Jika representasi ini diterapkan pada contoh kasus Mini Fasilkom pada subbab 3.2, panjang *chromosome* yang terjadi adalah 6 x 5 = 30. Ilustrasi dari representasi DRSRSK untuk *assignment* lengkap yang sesuai dengan contoh pada kasus Mini Fasilkom dapat dilihat pada Gambar 3.7. Tanda panah menunjukkan sambungan dari *chromosome*.

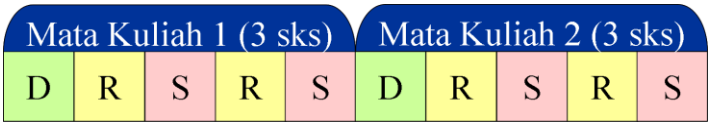


Gambar 3.7 Representasi DRSRSK pada Contoh Kasus Mini Fasilkom

3.3.4 Representasi DRSRS

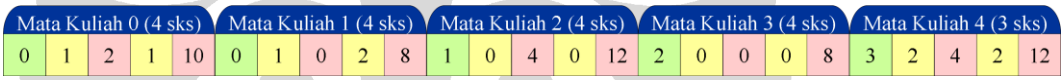
Representasi ini adalah modifikasi dari representasi DRSRSK, sehingga tetap menjamin HC1 terpenuhi. Pengertian dari representasi ini tidak jauh berbeda dengan representasi DRSRSK, hanya saja elemen K dihilangkan dan dianggap K

selalu sama dengan 0. Panjang *chromosome* untuk merepresentasikan jadwal dalam representasi ini adalah 5 x jumlah MK. Jika ada 2 MK, masing-masing berbobot 3 sks maka panjang *chromosome* adalah 5 x 2 = 10. Ilustrasi dari representasi DRSRS dapat dilihat pada Gambar 3.8.



Gambar 3.8 Representasi DRSRS

Jika representasi ini diterapkan pada contoh kasus Mini Fasilkom pada subbab 3.2, panjang *chromosome* yang terjadi adalah 5 x 5 = 25. Ilustrasi dari representasi DRSRS untuk *assignment* lengkap yang sesuai dengan contoh pada kasus Mini Fasilkom dapat dilihat pada Gambar 3.9.



Gambar 3.9 Representasi DRSRS pada Contoh Kasus Mini Fasilkom

3.3.5 Kekurangan dan Kelebihan Setiap Representasi

Berikut ini dibahas mengenai kekurangan dan kelebihan setiap representasi yang sudah dijelaskan pada subbab 3.3.1 - subbab 3.3.4.

1. Representasi MDRS

   Kelebihan:

   - Fleksibel, dalam arti representasi lain dapat dipetakan ke representasi MDRS, sehingga representasi ini bisa dijadikan *fenotype*.
   - *Search space* paling besar, mencakup representasi yang lain.

   Kekurangan:

   - Tidak menjamin jadwal semua MK sesuai dengan jumlah sksnya.
   - MK yang seharusnya ditawarkan bisa tidak terjadwal.
   - Memungkinkan 1 MK diajar oleh lebih dari 1 dosen, ruangan, dan slot. Hal ini tidak sepenuhnya merupakan kekurangan karena dalam kenyataan kasus Fasilkom, ada sedikit kuliah yang diajar oleh 2 dosen

secara bergantian, misalnya MK Grafkom di semester genap 2008/2009.

- Lebih lambat dari representasi yang lain dalam mencari solusi karena *search space* yang dimiliki paling besar.

2. Representasi DRS

   Kelebihan:

   - Menjamin jadwal semua MK sesuai dengan jumlah sksnya.
   - *Search space* besar, mencakup representasi DRSRSK dan DRSRS.

   Kekurangan:

   - Memungkinkan 1 MK diajar oleh lebih dari 1 dosen, ruangan, dan slot. Jumlah maksimum dosen, ruangan, dan slot sesuai dengan bobot sks MK tersebut. Jika Mk berbobot 4 sks, berarti bisa ada 4 dosen berbeda yang mengajar MK tersebut dengan 4 ruangan berbeda di 4 slot berbeda.
   - Lebih lambat dari representasi DRSRSK dan DRSRS dalam mencari solusi karena *search space* yang dimiliki lebih besar.

3. Representasi DRSRSK

   Kelebihan:

   - Menjamin jadwal semua MK sesuai dengan jumlah sksnya.
   - Menjamin 1 MK diajar oleh 1 dosen.
   - Menjamin 1 MK dipecah menjadi 2 kali pertemuan pada MK yang bobot sksnya lebih besar dari 2, sementara untuk MK yang bobot sksnya ≤ 2 diadakan 1 kali pertemuan. Dalam 1 kali pertemuan bisa 2 slot atau 1 slot di satu ruangan tetap.
   - Dapat menukar slot dengan mengubah nilai K dalam sekali mutasi.
   - Lebih cepat dari representasi MDRS dan DRS dalam mencari solusi karena *search space* yang dimiliki lebih kecil.

   Kekurangan:

   - Tidak memungkinkan kasus 1 MK diajar oleh 2 dosen.
   - *Search space* paling kecil.

4. Representasi DRSRS

   Kelebihan:

- Menjamin jadwal semua MK sesuai dengan jumlah sksnya.
- Menjamin 1 MK diajar oleh 1 dosen.
- Menjamin 1 MK dipecah menjadi 2 kali pertemuan pada MK yang bobot sksnya lebih besar dari 2, sementara untuk MK yang bobot sksnya ≤ 2 diadakan 1 kali pertemuan. Dalam 1 kali pertemuan bisa 2 slot atau 1 slot di satu ruangan tetap.
- Lebih cepat dari representasi MDRS dan DRS dalam mencari solusi karena *search space* yang dimiliki lebih kecil.
- Panjang *genome* paling pendek, paling hemat *space*.

Kekurangan:

- Tidak memungkinkan kasus 1 MK diajar oleh 2 dosen.
- *Search space* paling kecil.

3.4 Rancangan Fitness Function

Fitness function diperlukan untuk mengarahkan generasi ke arah yang lebih baik. *Fitness function* bisa terdiri dari 1 atau lebih fungsi objektif (*multiobjective*). Dalam masalah penjadwalan yang merupakan masalah yang *multiobjective*, 1 *constraint* dapat dipandang sebagai 1 fungsi objektif. Semakin sedikit pelanggaran terhadap suatu *constraint* semakin baik fungsi objektifnya dan akan semakin tinggi nilai *fitness* yang dicapai.

Dalam menentukan nilai *fitness* dari suatu individu ada 2 pendekatan, yaitu pendekatan *aggregation based* dan Pareto *based*. Pendekatan *aggregation based* menggunakan kombinasi linear dari fungsi-fungsi objektif. Sedangkan, pendekatan Pareto *based* menggunakan prinsip dominasi. Pada tugas akhir ini digunakan pendekatan Pareto *based* dengan algoritma SPEA2 (lihat subbab 2.3.2) dan pendekatan *aggregation based* dengan *fitness function* = (1/(1+Pinalti HC1) + 1/(1+Pinalti HC2) + … + 1/(1+Pinalti HC5) + 1/(1+Pinalti SC1) + 1/(1+Pinalti SC2) + … + 1/(1+Pinalti SC6)) / 11.

Setiap pelanggaran terhadap suatu *constraint* terdapat pinalti, semakin banyak pinalti semakin buruk nilai *fitness* yang diperoleh. Selanjutnya dibahas pinalti

untuk masing-masing *constraint* dan perhitungannya berdasarkan contoh *assignment* lengkap dari kasus Mini Fasilkom yang melanggar semua *constraint* (pada Gambar 3.10). Berdasarkan definisi variabel CSP, ada 19 baris *assignment* pada Gambar 3.10 yang masing-masing menyatakan assingnment pada *tuple* ke-i.

(0, 0, 1, 2),
(0, 0, 1, 3),
(0, 0, 1, 10),
(1, 0, 1, 11),
(1, 0, 1, 0),
(1, 0, 1, 2),
(1, 0, 2, 8),
(1, 0, 2, 9),
(1, 1, 0, 4),
(2, 1, 0, 2),
(2, 1, 0, 3),
(2, 1, 0, 13),
(3, 2, 1, 0),
(3, 2, 1, 1),
(3, 2, 0, 8),
(3, 2, 0, 7),
(4, 3, 2, 12),
(4, 3, 2, 4),
(4, 2, 2, 3)

Gambar 3.10 Contoh *Assignment* Lengkap Pada Kasus Mini Fasilkom yang Melanggar Semua *Constraint*

HC1 yaitu jadwal semua MK sesuai dengan jumlah sksnya. Semakin banyak MK yang tidak sesuai dengan jumlah sksnya semakin buruk jadwalnya. Semakin besar selisih sks antara MK yang tidak sesuai dengan MK yang benar, semakin buruk jadwalnya. Oleh karena itu, pinalti HC1 = total dari selisih sks MK yang tidak sesuai dengan jumlah sksnya. Pehitungan pinalti HC1 untuk contoh pada Gambar 3.10 yaitu MK 0 yang seharusnya 4 sks dijadwalkan hanya 3 sks, selisih ketidaksesuaian sks MK 0 adalah 1. Sementara MK 1 yang seharusnya 4 sks dijadwalkan 6 sks, selisih ketidaksesuaian sks MK 1 adalah 2. Sedangkan MK 2,

MK 3, dan MK 4 sudah sesuai. Jadi total dari selisih sks MK yang tidak sesuai dengan jumlah sksnya (pinalti HC1) = 1 + 2 + 0 + 0 + 0 = 3.

HC2 yaitu MK wajib di tingkat yang sama tidak bentrok. Semakin banyak slot di mana terdapat MK wajib setingkat yang bentrok, semakin buruk jadwalnya. Semakin banyak MK wajib setingkat yang bentrok di slot yang sama, semakin buruk jadwalnya. Oleh karena itu, pinalti HC2 = total jumlah MK wajib setingkat yang bentrok di slot yang sama. Pehitungan pinalti HC2 untuk contoh pada Gambar 3.10 yaitu perhatikan setiap slot, lalu cek apakah ada MK setingkat yang bentrok. Pada slot 0 MK 1 dan 3 bentrok, namun MK 1 dan 3 berbeda tingkat, sehingga tidak terjadi pelanggaran di slot 0. Pada slot 1 tidak ada MK yang bentrok. Pada slot 2 MK 0 dan 2 bentrok, padahal MK 0 dan 2 sama-sama wajib dan di tingkat 1, sehingga pelanggaran di slot 2 ada 1 buah. Pada slot 3 MK 0 dan 2 bentrok, sehingga pelanggaran di slot 3 ada 1 buah. Di slot lain tidak ternyata tidak ada lagi pelanggaran, sehingga total jumlah MK wajib setingkat yang bentrok di slot yang sama (pinalti HC2) = 1 + 1 = 2

HC3 yaitu ruang dan slot yang sama tidak diisi lebih dari satu MK. Oleh karena itu, pinalti HC3 = total dari jumlah MK yang mengisi ruang dan waktu yang sama. Pehitungan pinalti HC3 untuk contoh pada Gambar 3.10 yaitu perhatikan *pair* ruang dan slot. Jika ada *pair*(ruang, slot) yang muncul lebih dari 1 kali berarti terjadi pelanggaran. Dimulai dari *pair* di baris pertama yaitu *pair*(1, 2), ternyata pada baris ke 6 muncul *pair* yang sama, sehingga terjadi pelanggaran 1 kali. Dilanjutkan *pair* pada baris kedua yaitu *pair*(1, 3), ternyata tidak ada *pair* yang sama muncul di baris lain. Pencarian dilanjutkan hingga *pair* baris ke 19, ternyata pelanggaran lain hanya terjadi pada *pair*(1, 0) di baris ke 5 dan 13. Jadi total dari jumlah MK yang mengisi ruang dan waktu yang sama (pinalti HC3) = 1 + 1 = 2.

HC4 yaitu dosen tidak mengajar lebih dari satu MK pada slot yang sama. Oleh karena itu, pinalti HC4 = total dari jumlah MK berbeda yang diajar oleh dosen dan waktu yang sama. Pehitungan pinalti HC4 untuk contoh pada Gambar 3.10 yaitu perhatikan *pair* dosen dan slot. Jika ada *pair*(dosen, slot) yang muncul lebih

dari 1 kali berarti terjadi pelanggaran. Dimulai dari *pair* di baris pertama yaitu *pair*(0, 2), ternyata pada baris ke 6 muncul *pair* yang sama, sehingga terjadi pelanggaran 1 kali. Dilanjutkan *pair* pada baris kedua yaitu *pair*(0, 3), ternyata tidak ada *pair* yang sama muncul di baris lain. Pencarian dilanjutkan hingga *pair* baris ke 19, ternyata tidak ada yang melanggar. Jadi total dari jumlah MK berbeda yang diajar oleh dosen dan waktu yang sama (pinalti HC4) = 1.

HC5 yaitu jumlah pengikut suatu MK lebih kecil atau sama dengan kapasitas ruangannya. Oleh karena itu, pinalti HC5 = jumlah kelas yang kelebihan muatan. Pehitungan pinalti HC5 untuk contoh pada Gambar 3.10 yaitu dengan melihat kapasitas ruangan dan pengikut kuliah di setiap baris. Pada baris pertama MK 0 ditempatkan di ruangan 1, pengikut MK 0 ada 60 orang dan kapasitas ruangan 1 ada 65, sehingga tidak terjadi pelanggaran. Pencarian terus dilakukan pada baris selanjutnya. Pada baris 13 barulah ditemukan pelanggaran yaitu MK 3 yang memiliki pengikut 120 orang ditempatkan di ruang 1 yang berkapasitas 65, sehingga tidak muat. Pelanggaran lain hanya ditemukan pada baris ke 14. Jadi jumlah kelas yang kelebihan muatan (pinalti HC5) = 2.

SC1 yaitu dosen mengajar tidak melebihi maksimum beban sksnya. Oleh karena itu, pinalti SC1 = jumlah dosen yang kelebihan beban. Pehitungan pinalti SC1 untuk contoh pada Gambar 3.10 yaitu menjumlahkan total mengajar setiap dosen dan membandingkannya dengan maksimum beban dosen yang bersangkutan. Total mengajar dosen 0, 1, 2, dan 3 berturut-turut adalah 8, 4, 5, dan 2. Sementara beban maksimum dosen 0, 1, 2, dan 3 berturut-turut adalah 8, 4, 4, 3. Sehingga dosen 2 kelebihan beban. Jadi jumlah dosen yang kelebihan beban (pinalti SC1) = 1.

SC2 yaitu dosen hanya mengajar kuliah yang dia ekspertise. Oleh karena itu, pinalti SC2 = jumlah slot kuliah yang diajar oleh dosen yang bukan expertisenya. Pehitungan pinalti SC2 untuk contoh pada Gambar 3.10 yaitu untuk setiap baris dilihat apakah dosen mengajar MK yang dia ekspertise. Pada baris pertama dosen 0 mengajar MK 0, sementara ekspertise dosen 0 adalah [0, 1], berarti dosen 0

mengajar MK yang dia ekspertise, tidak terjadi pelanggaran. Demikian selanjutnya untuk setiap baris. Pada baris 9 dosen 1 mengajar MK 1, sementara ekspertise dosen 1 adalah [2], berarti dosen 1 mengajar MK yang bukan ekspertisenya, terjadi pelanggaran. Pelanggaran lain terjadi di baris 19 yaitu dosen 2 mengajar MK 4 yang bukan ekspertisenya. Jadi jumlah slot kuliah yang diajar oleh dosen yang bukan expertisenya (pinalti SC2) = 2.

SC3 yaitu dosen hanya mengajar pada waktu lowongnya. Oleh karena itu, pinalti SC3 = jumlah slot kuliah yang diajar oleh dosen tidak pada waktu lowongnya. Pehitungan pinalti SC3 untuk contoh pada Gambar 3.10 yaitu untuk setiap baris dilihat apakah dosen mengajar di waktu lowongnya. Pada baris pertama dosen 0 mengajar pada slot 2, sementara waktu lowong dosen 0 adalah [0, 1, 2, 3, 8, 9, 10, 11], berarti dosen 0 mengajar di waktu lowongnya, tidak terjadi pelanggaran. Demikian selanjutnya dicek untuk setiap baris. Pada baris 10 dosen 1 mengajar pada slot 2, sementara waktu lowong dosen 1 adalah [4, 5, 12, 13], berarti dosen 1 mengajar bukan pada waktu lowongnya, terjadi pelanggaran. Pelanggaran lain terjadi di baris 11, 16, dan 19. Jadi jumlah slot kuliah yang diajar oleh dosen tidak pada waktu lowongnya (pinalti SC3) = 4.

SC4 yaitu kuliah tidak terpotong makan siang. Oleh karena itu, pinalti SC4 = jumlah MK yang terpotong makan siang. Pehitungan pinalti SC4 untuk contoh pada Gambar 3.10 yaitu untuk setiap MK dilihat apakah terdapat slot kuliah pada jam perpotongan makan siang. Pada kasus ini terdapat pada slot 3, 4 (potongan pada hari Senin) dan 11, 12 (potongan pada hari Selasa). Dimulai dari MK 0, MK 0 diselenggarakan pada slot 2, 3, dan 10, dimana tidak mengandung 3 dan 4, hanya mengandung 3, juga tidak mengandung 11 dan 12. Sehingga tidak ada pelanggaran di MK 0. Demikian selanjutnya dicek untuk setiap MK. Ternyata, pelanggaran terjadi di MK 4 yang diselenggarakan pada slot 12, 4, dan 3, berarti mengandung 3 dan 4, sehingga melanggar SC4. Jadi jumlah MK yang terpotong makan siang (pinalti SC4) = 1.

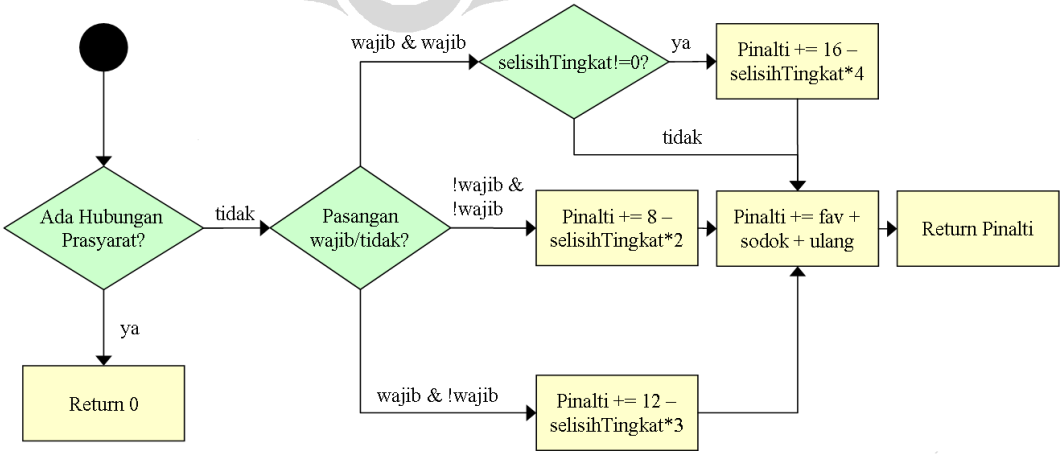
SC5 yaitu kuliah tidak terpotong ke hari berikutnya. Oleh karena itu, pinalti SC5 = jumlah MK yang terpotong ke hari berikutnya. Perhitungan pinalti SC5 untuk contoh pada Gambar 3.10 yaitu untuk setiap MK dilihat apakah terdapat slot kuliah pada jam perpotongan ke hari berikutnya. Pada kasus ini terdapat pada slot 7, 8 (potongan antara hari Senin dan Selasa). Dimulai dari MK 0, MK 0 diselenggarakan pada slot 2, 3, dan 10, dimana tidak mengandung 7 dan 8. Sehingga tidak ada pelanggaran di MK 0. Demikian selanjutnya dicek untuk setiap MK. Ternyata, pelanggaran terjadi di MK 3 yang diselenggarakan pada slot 0, 1, 8, dan 7, berarti mengandung 7 dan 8, sehingga melanggar SC5. Jadi jumlah MK yang terpotong ke hari berikutnya (pinalti SC5) = 1.

SC6 yaitu kuliah bentrok sebisa mungkin tidak mengganggu mahasiswa mengambil kuliah yang dia inginkan. Oleh karena itu, pinalti SC6 memperhitungkan hal-hal berikut.

1. Hubungan prasyarat. MK bentrok yang memiliki hubungan prasyarat tidak dikenakan pinalti karena tidak ada mahasiswa yang bisa mengambil kedua MK bersamaan karena MK yang satu merupakan prasyarat MK yang lain.
2. Wajib/tidaknya MK. MK wajib yang bentrok pinaltinya lebih besar dari MK tidak wajib karena setiap mahasiswa pasti akan mengmbil MK wajib dan biasanya pengikut MK wajib lebih banyak. Prinsipnya lebih sedikit mahasiswa yang komplain lebih baik. Jadi MK yang pengikutnya banyak lebih baik tidak bentrok, karena jika bentrok akan ada lebih banyak komplain.
3. Tingkat MK. MK bentrok dengan perbedaan tingkat yang kecil lebih besar pinaltinya daripada yang perbedaan tingkatnya besar karena mahasiswa cenderung hanya bisa mengambil MK yang perbedaan tingkatnya kecil secara bersama-sama. Sangat kecil kemungkinannya mahasiswa tingkat 1 mengambil MK tingkat 1 dan 4 sekaligus, tetapi lebih mungkin mengambil MK tingkat 1 dan 2 sekaligus.
4. Tingkat kefavoritan MK. Semakin tinggi tingkat kefavoritan suatu MK semakin banyak pengikutnya, sehingga pinaltinya lebih besar jika bentrok dengan MK lain.

5. Tingkat kecenderungan MK disodok. Semakin tinggi tingkat kecenderungan suatu MK disodok semakin banyak pengikutnya, sehingga pinaltinya lebih besar jika bentrok dengan MK lain.
6. Tingkat kecenderungan MK diulang. Semakin tinggi tingkat kecenderungan suatu MK diulang semakin banyak pengikutnya, sehingga pinaltinya lebih besar jika bentrok dengan MK lain.

Berdasarkan hal-hal yang diperhitungkan tersebut, dirancanglah perhitungan pinalti untuk SC6. Pinalti total merupakan jumlah pinalti yang dihitung untuk setiap pasangan MK yang bentrok di suatu slot. Perhitungan pinalti untuk setiap pasangan MK yang bentrok ditunjukkan pada Gambar 3.11, yaitu jika pasangan MK tersebut memiliki hubungan prasyarat pinalti = 0. Jika pasangan tidak memiliki hubungan prasyarat, dicek wajib tidaknya pasangan MK tersebut. Nilai awal pinalti = 0. Jika pasangan MK tersebut sama-sama MK wajib dan selisih tingkat pasangan MK tersebut tidak sama dengan 0, pinalti dihitung dengan rumus pinalti = pinalti + 16 - selisih tingkat pasangan MK tersebut * 4. Jika pasangan MK tersebut sama-sama bukan MK wajib pinalti dihitung dengan rumus pinalti = pinalti + 8 - selisih tingkat pasangan MK tersebut * 2. Sedangkan, jika pasangan MK tersebut salah satunya MK wajib dan satunya lagi bukan MK wajib, pinalti dihitung dengan rumus pinalti = pinalti + 12 - selisih tingkat pasangan MK tersebut * 3. Perancangan rumus ini selaras dengan hal-hal yang harus diperhitungkan pada pinalti SC6 poin 2 dan 3.



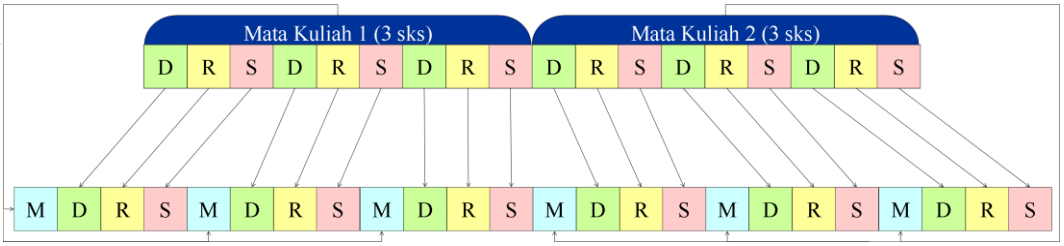
Gambar 3.11 Alur Perhitungan Pinalti SC6

Terakhir, pinalti ditambah dengan tingkat kefavoritan kedua pasang MK, tingkat kecenderungan kedua pasang MK disodok, dan tingkat kecenderungan kedua pasang MK diulang. Sehingga, perancangan pinalti selaras dengan hal-hal yang harus diperhitungkan pada pinalti SC6 poin 4, 5, dan 6.

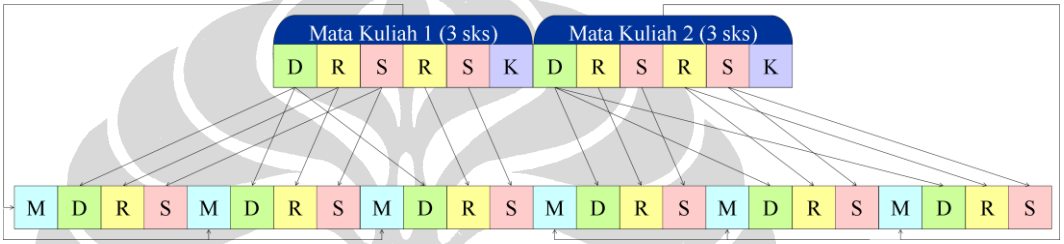
Pehitungan pinalti SC6 untuk contoh pada Gambar 3.10 yaitu untuk setiap slot dilihat apakah terdapat MK yang bentrok. Dimulai dari slot 0, terdapat MK yang bentrok yaitu MK 1 dan MK 3. Ternyata MK 1 dan MK 3 memiliki hubungan prasyarat, sehingga pinalti pada slot 0 tidak ada. Berlanjut ke slot 1, tidak ada MK yang bentrok di slot ini. Selanjutnya slot 2, terdapat 3 MK yang bentrok di slot ini yaitu MK 0,1, dan 2. Berarti akan dihitung pinalti untuk pasangan MK 0 dengan 1, 0 dengan 2, dan 1 dengan 2. Pertama-tama dihitung pinalti untuk pasangan MK 0 dengan 1 mengikuti aturan yang sudah dijelaskan sebelumnya. MK 0 dan MK 1 sama-sama merupakan MK wajib. MK 0 dan 1 sama-sama berada di tingkat 1, sehingga selisih tingkatnya sama dengan 0. Dengan demikian pinalti pasangan MK 0 dan MK 1 sama dengan penjumlahan tingkat kefavoritan kedua pasang MK, tingkat kecenderungan kedua pasang MK disodok, dan tingkat kecenderungan kedua pasang MK diulang = 1 (tingkat kefavoritan MK 0) + 1 (tingkat kefavoritan MK 1) + 0 (tingkat kecenderungan MK 0 disodok) + 0 (tingkat kecenderungan MK 1 disodok) + 1 (tingkat kecenderungan MK 0 diulang) + 1 (tingkat kecenderungan MK 1 diulang) = 4. Selanjutnya dihitung pinalti pasangan MK 0 dengan MK 2, yang hasilnya = 4 dengan mengikuti aturan yang sudah dijelaskan sebelumnya. Demikian selanjutnya dihitung pinalti untuk pasangan MK 1 dan MK 2, lalu dihitung pinalti di setiap slot dan akhirnya dihitung total semua pinalti.

Untuk menghitung *fitness function* digunakan representasi MDRS yang merupakan *fenotype*. Sementara representasi yang lain dipetakan ke representasi MDRS menggunakan prinsip *fenotype* dan *genotype*. Representasi yang lain adalah *genotype*, yaitu representasi DRS, DRSRSK, dan DRSRS. Ilustrasi

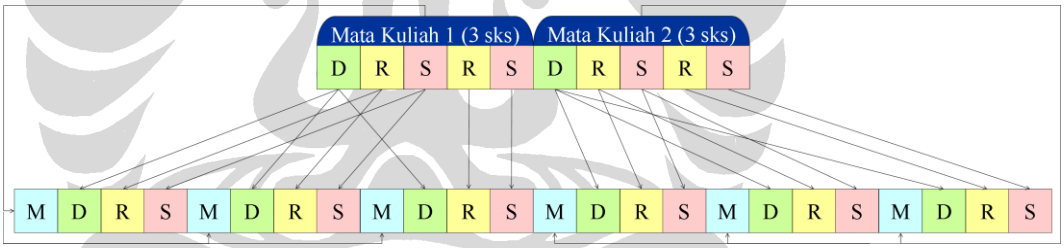
pemetaan dari representasi DRS ke MDRS, DRSRSK ke MDRS, dan DRSRS ke MDRS dapat dilihat pada Gambar 3.12-3.14.



Gambar 3.12 Pemetaan dari DRS ke MDRS



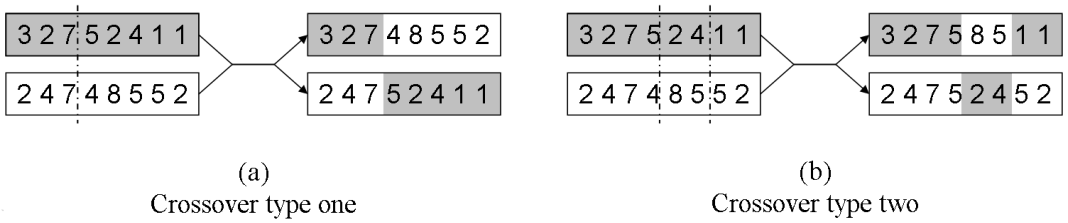
Gambar 3.13 Pemetaan dari DRSRSK ke MDRS



Gambar 3.14 Pemetaan dari DRSRS ke MDRS

3.5 Crossover dan Mutation

Crossover merupakan operasi genetik yang melibatkan dua individu sehingga anaknya merupakan kombinasi dari kedua inidividu induk. Ada dua tipe *crossover* yang digunakan pada tugas akhir ini dilihat dari jumlah *crossover point*, yaitu *crossover type one* dan *crossover type two*. *Crossover* yang digunakan pada contoh *problem* 8-*queens* (subbab 2.3) adalah *crossover type one*. Perbedaan *crossover type one* dan *two* pada *problem* 8-*queens* dapat dilihat pada Gambar 3.15.



Gambar 3.15 Perbedaan *Crossover Type One* dan *Two*

Mutation merupakan operasi genetik yang melibatkan satu individu dengan mengubah secara *random* gen dari individu tersebut. Bisanya operasi ini dilakukan dengan probabilitas kecil setelah operasi *crossover*. Ilustrasi *mutation* dapat dilihat pada Gambar 2.3 (e).

3.6 Rancangan Eksperimen

Variabel yang digunakan dalam eksperimen adalah sebagai berikut.

1. Ukuran *test case*: kecil (MK = 5, Dosen = 4, Ruang = 3, Slot = 16), sedang (MK = 20, Dosen = 20, Ruang = 17, Slot = 39), besar gasal (MK = 43, Dosen = 43, Ruang = 17, Slot = 39), dan besar genap (MK = 59, Dosen = 41, Ruang = 17, Slot = 39).
2. Algoritma *multiobjective*: SPEA2 dan *aggregation based* (AB).
3. Representasi: MDRS (R1), DRS (R2), DRSRSK (R3), dan DRSRS (R4).
4. GA Parameter: populasi (P) {50, 100, 200}, iterasi {5000}, *crossover rate* (CR) {1, 0.5}, *mutation rate* (MR) {0.01, 0.05, 0.1, 0.5}, *crossover type* (CT) {*one, two*}, dan *archive size* (A) {20%, 50%, 80%}.
5. *Constraint* Aktif: hanya HC (C1), HC + SC1 s/d SC5 (C2), dan semua HC + SC (C3).

Langkah eksperimen yang akan dijalankan adalah sebagai berikut.

1. Penentuan algoritma *multiobjective* dan representasi. Pada eksperimen ini dicoba semua kombinasi dari ukuran *test case*, algoritma, dan representasi dengan GA parameter: populasi 100, iterasi 5000, *crossover rate* 1, *mutation rate* 0.05, *crossover type one*, *archive size* 50%, dan *constraint* aktif: HC + SC1 s/d SC5. Masing-masing dicoba dengan 5 *random seed* dan dirata-rata. Dipilih 5 kali percobaan dengan *random seed* yang berbeda karena 5 tidak

terlalu kecil dan terlalu besar untuk menghindari faktor kebetulan dalam waktu yang tidak terlalu lama. Tujuan eksperimen ini adalah untuk memilih algoritma dan representasi terbaik untuk dipakai pada eksperimen selanjutnya.

2. Performa GA sebelum parameter dioptimasi. Pada eksperimen ini dicoba semua kombinasi dari ukuran *test case* dan *constraint* aktif dengan algoritma terbaik, representasi terbaik, GA parameter: populasi 100, iterasi 5000, *crossover rate* 1, *mutation rate* 0.05, *crossover type one*, *archive size* 50%. Masing-masing dicoba dengan 5 *random seed* dan dirata-rata. Tujuan dari eksperimen ini adalah untuk mengetahui performa dari GA dengan parameter yang belum dioptimasi.
3. Penentuan parameter ukuran populasi dan *archive size*. Pada eksperimen ini dicoba semua kombinasi dari ukuran *test case*, GA parameter populasi, dan GA parameter *archive size* dengan algoritma terbaik, representasi terbaik, GA parameter: iterasi 5000, *crossover rate* 1, *mutation rate* 0.05, *crossover type one*. Masing-masing dicoba dengan 5 *random seed* dan dirata-rata. Tujuan dari eksperimen ini adalah untuk mencari parameter populasi dan *archive size* terbaik.
4. Penentuan parameter operator genetik. Pada eksperimen ini dicoba semua kombinasi dari ukuran *test case*, GA parameter *mutation rate*, GA parameter *crossover rate*, dan GA parameter *crossover type* dengan algoritma terbaik, representasi terbaik, GA parameter: populasi terbaik, iterasi 5000, *archive size* terbaik. Masing-masing dicoba dengan 5 *random seed* dan dirata-rata. Tujuan dari eksperimen ini adalah untuk mencari *crossover rate* terbaik, *crossover type* terbaik, dan *mutation rate* terbaik.
5. Performa GA sesudah parameter dioptimasi. Pada eksperimen ini dicoba semua kombinasi dari ukuran *test case* dan *constraint* aktif dengan algoritma terbaik, representasi terbaik, dan GA parameter terbaik. Masing-masing dicoba dengan 5 *random seed* dan dirata-rata. Tujuan dari eksperimen ini adalah untuk mengetahui performa dari GA dengan parameter yang sudah dioptimasi.

BAB 4 IMPLEMENTASI

Pada bab ini dibahas implementasi berdasarkan perancangan yang sudah dilakukan pada bab 3 yang meliputi implementasi parameter GA (subbab 4.1), implementasi *fitness function* (subbab 4.2), *input* & *output* (subbab 4.3), dan implementasi eksperimen (subbab 4.4).

4.1 Implementasi Parameter GA

Dalam implementasi pada tugas akhir ini digunakan bahasa pemrograman java dan dengan SDK Eclipse. *Library* yang digunakan untuk GA yaitu ECJ (*Evolutionary Computation for Java*). Penggunaan *library* ini didasarkan atas pertimbangan: tersediannya algoritma *multiobjective* SPEA2, penggunaannya yang menggunakan parameter *file* sehingga lebih mudah diatur dari luar *source code*, dan ECJ adalah *library* GA yang paling unggul berdasarkan *benchmark*.

Dengan parameter *file* yang memiliki *ekstension* (.param), GA parameter bisa diubah dari luar *source code*. Contoh GA parameter antara lain jumlah populasi, banyak iterasi (*generation*), *crossover rate*, *mutation rate*, *crossover type*, *archive size*, *random seed*, *selection method*, panjang *chromosome*, dan *domain genome*. Berikut ini adalah contoh parameter yang bisa diatur diikuti dengan *source code* yang sesuai.

Jumlah populasi

```
pop.subpop.0.size = 100
```

Banyak iterasi (*generation*)

```
generations = 1000
```

Crossover rate

```
pop.subpop.0.species.crossover-prob = 1.0
```

Mutation rate

```
pop.subpop.0.species.mutation-prob = 0.05
```

Crossover type

```
pop.subpop.0.species.crossover-type = one
```

Archive size

```
pop.subpop.0.archive-size  = 50
```

Random seed

```
seed.0  = 4357
```

Selection method

```
exch.subpop.0.select = ec.select.TournamentSelection
```

Evaluasi *problem*

```
eval.problem  = penjadwalanSPEA2R2.PenjadwalanSPEA2R2
```

Jenis individu

```
pop.subpop.0.species.ind = ec.vector.IntegerVectorIndividual
```

Panjang *chromosome*

```
pop.subpop.0.species.genome-size= 57
```

Domain genome

```
pop.subpop.0.species.min-gene = 1
pop.subpop.0.species.max-gene = 2
pop.subpop.0.species.min-gene.0 = 0
pop.subpop.0.species.max-gene.0 = 3
pop.subpop.0.species.min-gene.1 = 0
pop.subpop.0.species.max-gene.1 = 2
```

Breeder

```
breed   = ec.multiobjective.spea2.SPEA2Breeder
```

Evaluator

```
eval    = ec.multiobjective.spea2.SPEA2Evaluator
```

Populasi

```
pop.subpop.0  = ec.multiobjective.spea2.SPEA2Subpopulation
```

Fitness function

```
pop.subpop.0.species.fitness        =
ec.multiobjective.spea2.SPEA2MultiObjectiveFitness
```

Contoh parameter *file* secara utuh dapat dilihat pada Lampiran B.

4.2 Implementasi Fitness Function

Implementasi *fitness function* ada di Fenotype.java. Perhitungan *fitness function* berdasarkan representasi MDRS yang merupakan representasi *fenotype*. Setiap *constraint* dibuat satu *method* tersendiri. Pada Gambar 4.1-4.11 ditamplkan *pseudocode* masing-masing *method* secara berturut-turut dari HC1, HC2, HC3, HC4, HC5, SC1, SC2, SC3, SC4, SC5, SC6. Pada Gambar 4.12 ditampilkan *pseudocode* hitung pinalti yang digunakan pada *pseudocode* SC6.

```
function hc1Fitness()
    hc1ViolationCount = 0
    for xi = (mi di ri si) in X
        mkCounter[mi]++
    for i from 1 to size(MK)
        hc1ViolationCount += selisih(jumlahsks(mi),mkCounter[mi])
    return 1/(1+hc1ViolationCount)
```

Gambar 4.1 *Pseudocode Fitness* HC1

```
function hc2Fitness()
    hc2ViolationCount = 0
    for t = tingkat MK
        slot = {}
        for xi = (mi di ri si) in X
            if tingkat(mi) = t and wajib(mi)
                if si ∈ slot
                    hc2ViolationCount++
                else
                    add si to slot
    return 1/(1+hc2ViolationCount)
}
```

Gambar 4.2 *Pseudocode Fitness* HC2

```
function hc3Fitness()
    hc3ViolationCount = 0
    for xi = (mi di ri si) in X
        ruangSlot[i] = "R" + ri + "S" + si
    hc3ViolationCount = getJumlahSama(ruangSlot)
    return 1/(1+hc3ViolationCount)
```

Gambar 4.3 *Pseudocode Fitness* HC3

```
function hc4Fitness()
    hc4ViolationCount = 0
    for xi = (mi di ri si) in X
        dosenSlot[i] = "D" + ri + "S" + si
    hc4ViolationCount = getJumlahSama(dosenSlot)
    return  1/(1+hc4ViolationCount)
```

Gambar 4.4 *Pseudocode Fitness* HC4

```
function hc5Fitness()
    hc5ViolationCount = 0
    for xi = (mi di ri si) in X
        if jumlahPengikut(mi) > kapasitas(ri)
            hc5ViolationCount++
    return 1/(1+hc5ViolationCount)
```

Gambar 4.5 *Pseudocode Fitness* HC5

```
function sc1Fitness()
    sc1ViolationCount = 0
    for xi = (mi di ri si) in X
        sksDosen[di]++
    for i from 1 to size(Dosen)
        if maksimumSks(i) < sksDosen[i]
            sc1ViolationCount++
    return 1/(1+sc1ViolationCount)
```

Gambar 4.6 *Pseudocode Fitness* SC1

```
function sc2Fitness()
    sc2ViolationCount = 0
    for xi = (mi di ri si) in X
        if mi ∉ expertiseMk(di)
            sc2ViolationCount++
    return 1/(1+sc2ViolationCount)
```

Gambar 4.7 *Pseudocode Fitness* SC2

```
function sc3Fitness()
    sc3ViolationCount = 0
    for xi = (mi di ri si) in X
        if ri ∉ waktuLuang(di)
            sc3ViolationCount++
    return 1/(1+sc3ViolationCount)
```

Gambar 4.8 *Pseudocode Fitness* SC3

```
function sc4Fitness()
    sc4ViolationCount = 0
    for xi = (mi di ri si) in X
        add si to mkSlot[mi] //mkSlot mi adalah sorted set
    for i from 1 to size(MK)
        tmp = mkSlot[i]
        for j from 1 to size(tmp)-1
            if tmp[j+1] – tmp[j] = 1 //j and j+1 adalah slot yang berurutan
                if sebelumJam12(tmp[j]) = true and sebelumJam12(tmp[j+1]) = false
                    sc4ViolationCount++
    return 1/(1+sc4ViolationCount)
```

Gambar 4.9 *Pseudocode Fitness* SC4

```
function sc5Fitness()
    sc5ViolationCount = 0
    for xi = (mi di ri si) in X
        add si to mkSlot[mi] //mkSlot mi adalah sorted set
    for i from 1 to size(MK)
        tmp = mkSlot[i]
        for j from 1 to size(tmp)-1
            if tmp[j+1] – tmp[j] = 1 //j and j+1 adalah slot yang berurutan
                if sebelumJam12(tmp[j]) = false and sebelumJam12(tmp[j+1]) = true
                    sc5ViolationCount++
    return 1/(1+sc5ViolationCount)
```

Gambar 4.10 *Pseudocode Fitness* SC5

```
function sc6Fitness()
    sc6ViolationCount = 0
    for xi = (mi di ri si) in X
        add mi to mkInSlot[si]
    penalti = 0
    for i from 1 to size(Slot)
        if size(mkInSlot[i])  > 1
            for j from 1 to size(mkInSlot[i])-1
                for k from j+1 to size(mkInSlot[i])
                    penalti += hitungPenalti(mkInSlot[i][j],mkInSlot[i][k])
    sc6ViolationCount = penalti
    return 1/(1+sc6ViolationCount)
```

Gambar 4.11 *Pseudocode Fitness* SC6

Untuk memudahkan melihat apa yang melanggar *constraint*, tidak hanya skor pinaltinya, dibuatlah laporan pelanggaran *constraint*. Implementasinya tidak jauh berbeda dengan menghitung pinalti, hanya saja ada sedikit modifikasi karena tujuannya untuk melaporkan apa yang melanggar, bukan berapa yang melanggar. Laporan ini dipanggil hanya sekali setelah mendapat individu yang terbaik dan individu ini yang dievaluasi oleh laporan. Kode laporan pelanggaran *constraint* ini ada di MyStatistic.java. MyStatistic.java merupakan anak dari *class* Statistic yang memiliki *method* yang dipanggil pada *event-event* tertentu, salah satunya saat evolusi berakhir.

```
function hitungPenalti(mkA, mkB)
   int hasil = 0
   if adaHubunganPrasyarat(mkA,mkB)
      return 0
   else
      selisihTingkat = absolute(tingkat(mkA) - tingkat(mkB))
      if wajib(mkA) and wajib(mkB)
         if selisihTingkat != 0
            hasil += 16 - selisihTingkat * 4
      else if !wajib(mkA) and !wajib(mkB)
         if selisihTingkat = 0
            hasil += 8
         else
            hasil += 8 - selisihTingkat * 2
      else
         if selisihTingkat = 0
            hasil += 12
         else
            hasil += 12 - selisihTingkat * 3
      hasil += favorit(mkA)
      hasil += favorit(mkB)
      hasil += sodok(mkA)
      hasil += sodok(mkB)
      hasil += ulang(mkA)
      hasil += ulang(mkB)
   return hasil
```

Gambar 4.12 *Pseudocode* Hitung Pinalti Untuk SC6

4.3 Input & Output

Input dari program adalah *file-file* yang berisi informasi tentang MK, dosen, ruang, dan slot. Masing-masing informasi dimuat dalam *file* teks terpisah. Selanjutnya dibahas mengenai masing-masing *file input*, keterangannya, dan contoh *file* yang bersangkutan.

Informasi tentang MK disimpan dalam 2 *file input*, yaitu mk.txt dan graphmk.txt. Contoh *file* mk.txt dapat dilihat pada Gambar 4.1 dan contoh *file* graphmk.txt

dapat dilihat pada Gambar 4.2. Arti dari mk.txt mirip dengan arti pada Tabel 3.1, yaitu kode MK, nama MK, bobot sks, jumlah pengikut, tingkat, wajib atau tidak, tingkat favorit, tingkat sodok, dan tingkat ulang. Sementara arti dari graphmk.txt yaitu kode MK prasarat dan kode MK yang memiliki prasyarat. Contoh baris pertama dapat dibaca MK kode 0 merupakan prasyarat dari MK kode 3, atau kalau dilihat nama MK dari mk.txt dapat dibaca IKI10820 Dasar-Dasar Pemrograman merupakan prasyarat dari IKI20100 Struktur Data & Algoritma.

```
0,IKI10820 Dasar-Dasar Pemrograman A,4,60,1,true,1,1,1
1,IKI10820 Dasar-Dasar Pemrograman B,4,60,1,true,1,1,1
2,IKI10201 Pengantar Sistem Digital,4,120,1,true,1,1,1
3,IKI20100 Struktur Data & Algoritma,4,120,2,true,1,1,1
4,IKI20200 Organisasi Sistem Komputer,3,60,2,true,1,1,1
```

Gambar 4.13 Contoh *File* mk.txt

```
0,3
1,3
2,4
```

Gambar 4.14 Contoh *File* graphmk.txt

Informasi tentang dosen disimpan dalam dosen.txt. Contoh *file* dosen.txt dapat dilihat pada Gambar 4.3. Arti dari dosen.txt mirip dengan arti pada Tabel 3.2, yaitu kode dosen, nama dosen, *list* kode MK yang dosen tersebut ekspertise, *list* kode slot yang dosen tersebut lowong, dan maksimum sks yang bisa dibebankan terhadap dosen tersebut.

```
0,Halsen,[0 1],[0 1 2 3 8 9 10 11],8
1,Kivin,[2],[4 5 12 13],4
2,Rus,[3],[0 1 8 9],4
3,Yuba,[4],[4 5 12],3
```

Gambar 4.15 Contoh *File* dosen.txt

```
0,R 2102,130
1,R 2301,65
2,R 2302,75
```

Gambar 4.16 Contoh *File* ruang.txt

Informasi tentang ruangan disimpan dalam ruang.txt. Contoh *file* ruang.txt dapat dilihat pada Gambar 4.4. Arti dari ruang.txt mirip dengan arti pada Tabel 3.3, yaitu kode ruangan, nama ruangan, dan kapasitas ruangan. Sementara informasi

tentang slot disimpan dalam slot.txt. Contoh *file* slot.txt dapat dilihat pada Gambar 4.5. Arti dari slot.txt mirip dengan arti pada Tabel 3.4, yaitu kode slot, hari dan jam slot tersebut berlangsung, dan *flag* yang menandakan apakah slot tersebut berlangsung sebelum makan siang.

```
0,Senin(08.00-09.00),true
1,Senin(09.00-10.00),true
2,Senin(10.00-11.00),true
3,Senin(11.00-12.00),true
4,Senin(13.00-14.00),false
5,Senin(14.00-15.00),false
6,Senin(15.00-16.00),false
7,Senin(16.00-17.00),false
8,Selasa(08.00-09.00),true
9,Selasa(09.00-10.00),true
10,Selasa(10.00-11.00),true
11,Selasa(11.00-12.00),true
12,Selasa(13.00-14.00),false
13,Selasa(14.00-15.00),false
14,Selasa(15.00-16.00),false
15,Selasa(16.00-17.00),false
```

Gambar 4.17 Contoh *File* slot.txt

Untuk merepresentasikan informasi *input* di atas dibuatlah *class* MataKuliah, *class* Dosen, *class* Ruang, dan *class* Slot. Lalu, untuk membaca *file-file input* dibuatlah *class* Data. *Class* ini sekaligus merepresentasikan Data *input* secara keseluruhan. Di dalam *class* ini dibuat *instance* dari *class* MataKuliah, Dosen, Ruang, dan Slot yang berupa *list*. Semua *class* tersebut digabungkan dalam satu *package* bernama *database*.

Output dari implementasi yang dibuat adalah *file* teks yang berisi jadwal yang diambil dari individu terbaik yang pernah muncul sepanjang generasi. Format dari jadwal tidak menggunakan kode, melainkan menggunakan nama sehingga lebih mudah dimengerti. Selain itu, ditampilkan juga laporan pelanggaran *constraint* agar memudahkan untuk melihat pelanggaran yang terdapat pada jadwal. Contoh dari *file output* dapat dilihat pada Lampiran C.

4.4 Implementasi Eksperimen

Variabel eksperimen dan cara yang digunakan sebagai implementasi dari pengubahan variabel tersebut dilakukan dengan cara berikut.

1. Ukuran *test case* (kecil, sedang, besar gasal, besar genap). Untuk mengubah ukuran *test case* dilakukan perubahan pada *source code* di PenjadwalanXY.java. XY berisi informasi algoritma dan representasi yang digunakan. Intinya adalah mengubah *path input file* dari *source code*.
2. Algoritma *multiobjective* (SPEA2, AB). Untuk mengubah algoritma yang digunakan, dilakukan perubahan di *run argument*. Contoh ketika menjalankan algoritma SPEA2 dengan representasi DRS:
   -file src/penjadwalanSPEA2R2/SPEA2besarGenapR2.params
3. Representasi (MDRS, DRS, DRSRSK, DRSRS). Untuk mengubah representasi yang digunakan, dilakukan perubahan di *run argument* seperti contoh di atas.
4. GA Parameter (populasi, iterasi, *crossover rate*, *mutation rate*, *crossover type*, *archive size*). Untuk mengubah GA parameter yang digunakan, dilakukan perubahan di parameter *file* atau bisa dilakukan di *run argument*.
5. *Constraint* Aktif (C1, C2, C3). Untuk mengubah GA parameter yang digunakan, dilakukan perubahan di *source code* di bagian yang mengitung *fitness function*, kemudian dilakukan *compile*.

Agar eksperimen bisa ditinggal, dibuatlah *batch file* yang fungsinya melakukan eksekusi secara berurut sesuai *setting*. Contoh *batch file* dapat dilihat pada Lampiran D.

BAB 5 HASIL DAN PEMBAHASAN

Pada bab ini dibahas hasil dari eksperimen yang dirancang pada subbab 3.6. Terdapat 5 eksperimen yang dijalankan, masing-masing hasil eksperimen dibahas pada subbab yang berbeda. Hasil eksperimen penentuan algoritma *multiobjective* dan representasi dibahas pada subbab 5.1, hasil eksperimen performa GA sebelum parameter dioptimasi dibahas pada subbab 5.2, hasil eksperimen penentuan parameter ukuran populasi dan *archive size* dibahas pada subbab 5.3, hasil eksperimen penentuan parameter operator genetik dibahas pada subbab 5.4, dan hasil eksperimen performa GA sesudah parameter dioptimasi dibahas pada subbab 5.5.

Tidak semua grafik hasil eksperimen ditampilkan pada subbab 5.1 - 5.5. Rekap grafik yang lengkap pada setiap eksperimen dapat dilihat pada Lampiran A. Beberapa singkatan dipakai pada grafik untuk menyatakan variabel eksperimen seperti pada subbab 3.6. Daftar singkatan yang digunakan dan artinya dijelaskan pada Tabel 5.1.

Grafik yang ditampilkan adalah grafik konvergensi yang menunjukkan *progress* nilai *fitness function* selama iterasi GA berlangsung. Sumbu x menyatakan iterasi dari 0 - 5000 (bilangan bulat) dan sumbu y menyatakan nilai *fitness function* yang nilainya berkisar antara 0 - 1 (bilangan real). Nilai *fitness* yang ditampilkan adalah rata-rata dari nilai *fitness* individu terbaik dari populasi pada iterasi tertentu dari 5 *random seed*. Perhitungan nilai *fitness* individu bergantung dari *constraint* apa yang aktif. Selanjutnya dibahas cara menghitung *fitness function* untuk eksperimen yang menggunakan algoritma *aggregation based* dan algoritma SPEA2.

Tabel 5.1 Daftar Singkatan Grafik

Singkatan	Arti
A	*archive size*
A20	ukuran *archive* 20% dari jumlah populasi
A50	ukuran *archive* 50% dari jumlah populasi
A80	ukuran *archive* 80% dari jumlah populasi
AB	*aggregation based*
AVG	*average* (rata-rata)
C1	*constraint* aktif: hanya HC
C2	*constraint* aktif: HC + SC1 s/d SC5
C3	*constraint* aktif: semua HC dan SC
CR	*crossover rate*
CR05	*crossover* dengan probabilitas 0.5
CR1	*crossover* dengan probabilitas 1
CT	*crossover type*
CTone	*crossover* dengan 1 *crossover point*
CTtwo	*crossover* dengan 2 *crossover point*
MR	*mutation rate*
MR00005	*mutation* dengan probabilitas 0.0005
MR0001	*mutation* dengan probabilitas 0.001
MR00025	*mutation* dengan probabilitas 0.0025
MR0005	*mutation* dengan probabilitas 0.005
MR001	*mutation* dengan probabilitas 0.01
MR005	*mutation* dengan probabilitas 0.05
MR01	*mutation* dengan probabilitas 0.1
MR05	*mutation* dengan probabilitas 0.5
P	populasi
P100	jumlah populasi 100
P200	jumlah populasi 200
P50	jumlah populasi 50
R	representasi
R1	representasi MDRS
R2	representasi DRS
R3	representasi DRSRSK
R4	representasi DRSRS

Cara menghitung nilai *fitness function* untuk eksperimen yang menggunakan algoritma *multiobjective aggregation based* dengan *constraint* aktif:

- C1, yaitu *fitness function* = (1/(1+Pinalti HC1) + 1/(1+Pinalti HC2) + … + 1/(1+Pinalti HC5)) / 5,

- C2, yaitu *fitness function* = (1/(1+Pinalti HC1) + 1/(1+Pinalti HC2) + … + 1/(1+Pinalti HC5) + 1/(1+Pinalti SC1) + 1/(1+Pinalti SC2) + … + 1/(1+Pinalti SC5)) / 10,
- C3, yaitu *fitness function* = (1/(1+Pinalti HC1) + 1/(1+Pinalti HC2) + … + 1/(1+Pinalti HC5) + 1/(1+Pinalti SC1) + 1/(1+Pinalti SC2) + … + 1/(1+Pinalti SC6)) / 11.

Algoritma SPEA2 memiliki cara sendiri dalam menghitung *fitness function* (lihat subbab 2.3.2), *fitness function* inilah yang digunakan dalam *selection*, sehingga menghasilkan generasi yang sesuai dengan *fitness function* tersebut. Namun, nilai *fitness function* ini tidak dapat dibandingkan dengan nilai *fitness function* pada algoritma *aggregation based*. Oleh karena itu, dihitung lagi nilai *fitness function* setiap inidividu pada SPEA2 berdasarkan *fitness function aggregation based* agar bisa dibandingkan, namun tidak mempengaruhi generasi pada SPEA2, hanya sebagai alat untuk membandingkan. Jadi cara menghitung nilai *fitness function* untuk eksperimen yang menggunakan algoritma *multiobjective* SPEA2 sama dengan algoritma *multiobjective aggregation based*.

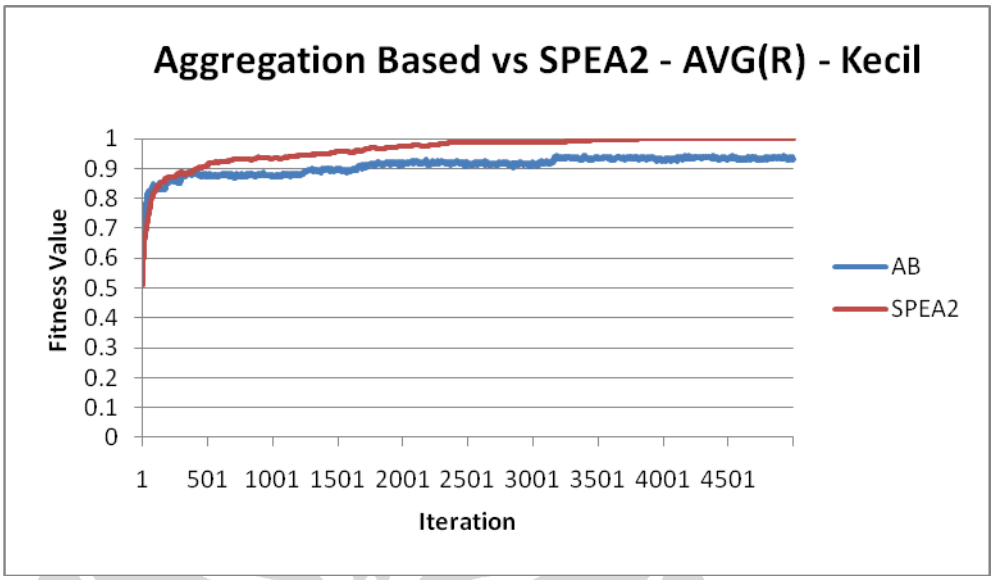
5.1 Hasil Penentuan Algoritma Multiobjective dan Representasi

Dalam eksperimen penentuan algoritma *multiobjective* dan representasi, yang dicoba adalah semua kombinasi dari ukuran *test case* (kecil, sedang, besar gasal, besar genap), algoritma (AB, SPEA2), dan representasi (R1, R2, R3, R4) dengan GA parameter: populasi 100, iterasi 5000, *crossover rate* 1, *mutation rate* 0.05, *crossover type one*, *archive size* 50%, dan *constraint* aktif: HC + SC1 s/d SC5. Masing-masing dicoba dengan 5 *random seed* dan dirata-rata. Tujuan eksperimen ini adalah untuk memilih algoritma dan representasi terbaik untuk dipakai pada eksperimen selanjutnya.

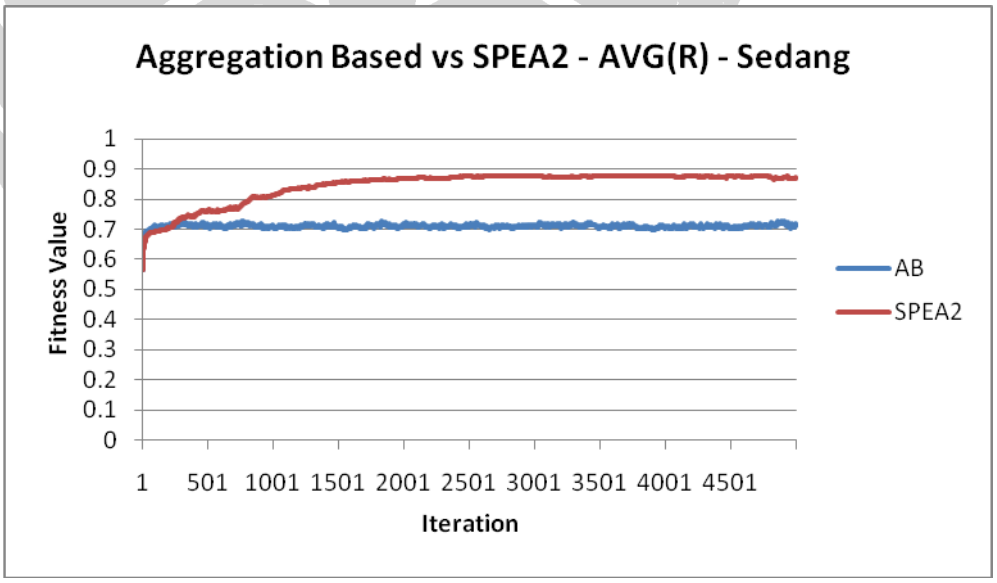
5.1.1 Penentuan Algoritma Multiobjective: AB vs SPEA2

Berikut ini gambar grafik yang menampilkan perbandingan antara algoritma *aggregation based* dan SPEA2 dengan merata-ratakan hasil di keempat representasi (R1, R2, R3, R4) pada *test case* kecil (Gambar 5.1), sedang (Gambar

5.2), besar gasal (Gambar 5.3), dan besar genap (Gambar 5.4) untuk menentukan algoritma terbaik.



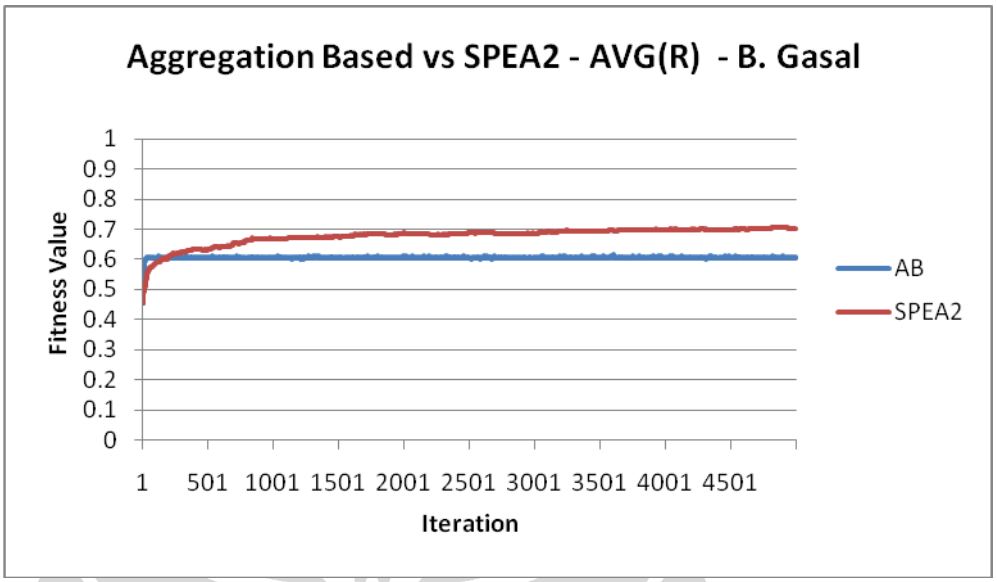
Gambar 5.1 *Aggregation Based* vs SPEA2 – Avg(R) - Kecil



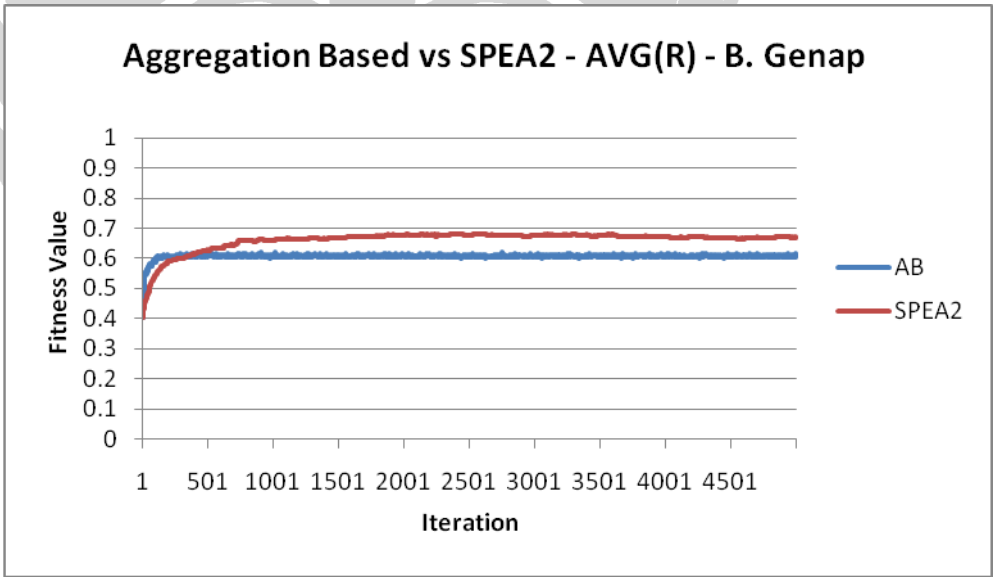
Gambar 5.2 *Aggregation Based* vs SPEA2 – Avg(R) - Sedang

Dari Gambar 5.1 dapat dilihat bahwa untuk *test case* kecil SPEA2 dapat menemukan solusi optimal (ditandai dengan nilai *fitness function* = 1), sedangkan AB tidak bisa menemukan solusi optimal, melainkan hanya dapat mencapai nilai *fitness* maksimal 0.93. Lebih lagi, pada Gambar 5.2, 5.3, dan 5.4 dapat dilihat SPEA2 lebih unggul dari AB walaupun kalah pada iterasi awal. Kedua algoritma,

baik SPEA2 maupun AB tidak dapat menemukan solusi optimal pada *test case* sedang, besar gasal, dan besar genap.



Gambar 5.3 *Aggregation Based* vs SPEA2 – Avg(R) - Besar Gasal



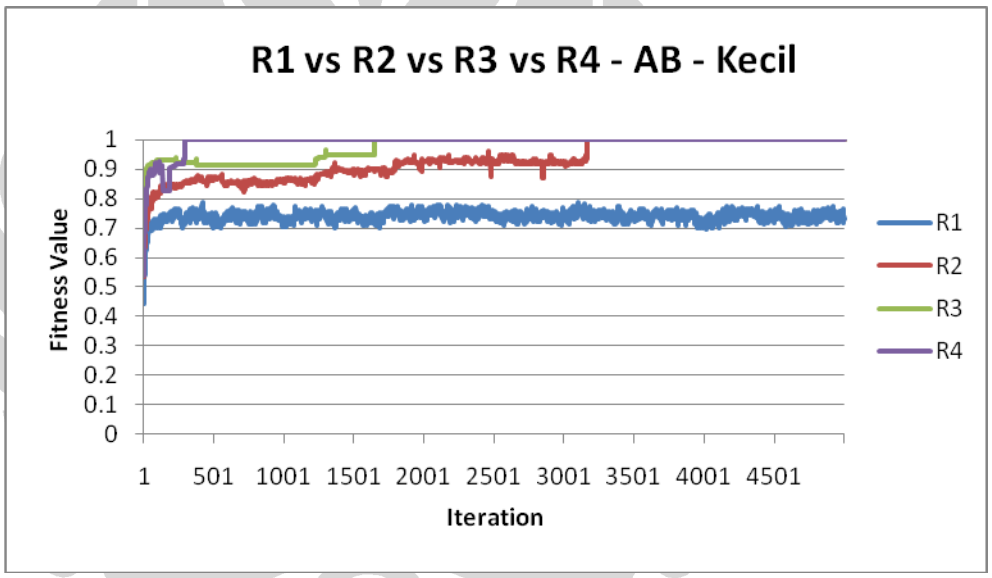
Gambar 5.4 *Aggregation Based* vs SPEA2 – Avg(R) - Besar Genap

Dari hasil di atas dapat ditarik kesimpulan bahwa algoritma SPEA2 lebih baik dibandingkan dengan *aggregation based*. Hal ini karena algoritma SPEA2 memaksimalkan semua fungsi objektif, sementara algoritma *aggregation based* memaksimalkan agregasi dari nilai semua fungsi objektif. Alasan lain SPEA2 unggul karena algoritma ini mempertahankan keragaman individu, tidak ada satu

fungsi objektif pun yang diabaikan walaupun nilainya kecil. Sementara pada algoritma *aggregation based* jika ada satu fungsi objektif yang nilainya kecil dan tidak terlalu berpengaruh terhadap total nilai *fitness function*, besar kemungkinan fungsi objektif ini diabaikan.

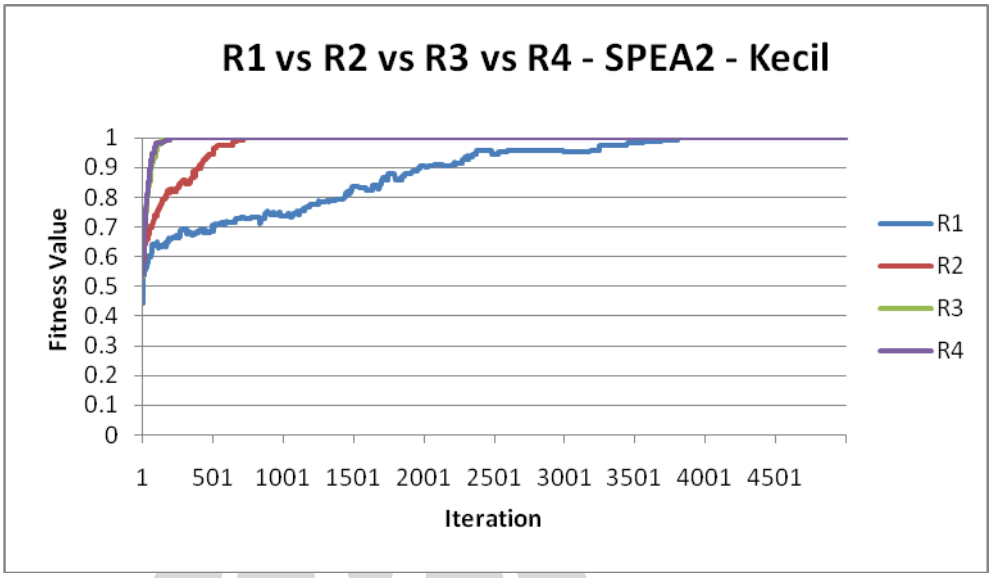
5.1.2 Penentuan Representasi: R1 vs R2 vs R3 vs R4

Berikut ini ditampilkan gambar grafik perbandingan representasi pada algoritma AB dan SPEA2 pada *test case* kecil (Gambar 5.5 dan Gambar 5.6), sedang (Gambar 5.7 dan Gambar 5.8). besar gasal (Gambar 5.9 dan Gambar 5.10) dan besar genap (Gambar 5.11 dan Gambar 5.12) untuk menentukan representasi terbaik.

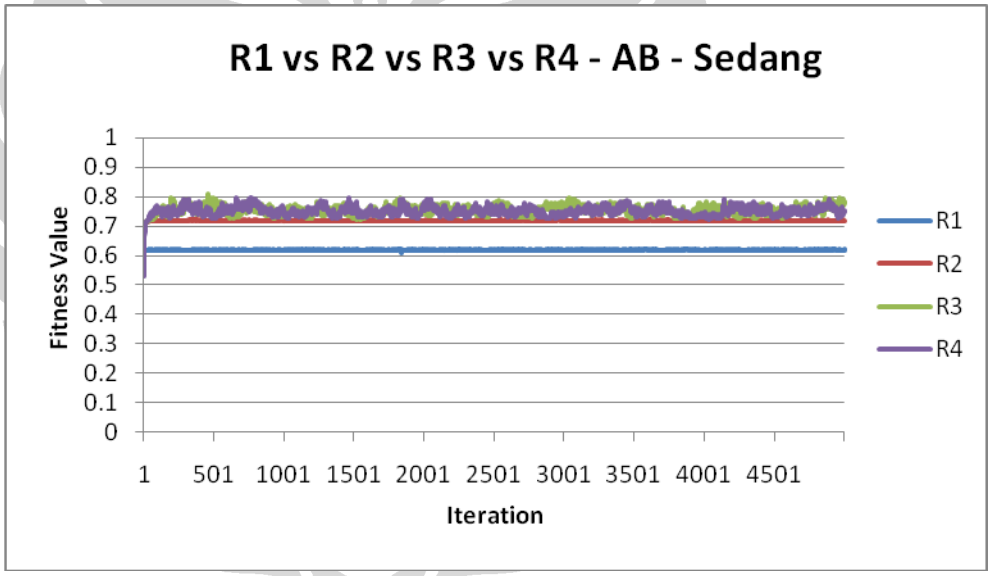


Gambar 5.5 R1 vs R2 vs R3 vs R4 - AB - Kecil

Dari Gambar 5.5 terlihat bahwa R2, R3, R4 dapat menemukan solusi optimal, sedangkan R1 tidak bisa. Urutan kecepatan representasi dalam menemukan solusi optimal yaitu R4 (sekitar 400 iterasi), R3 (sekitar 1700 iterasi), dan yang terakhir R2 (sekitar 3300 iterasi). Sementara, dari Gambar 5.6 terlihat bahwa semua representasi dapat menemukan solusi optimal dengan urutan kecepatan R3 dan R4 hampir bersamaan (sekitar 300 iterasi), lalu R2 (sekitar 800 iterasi), dan terakhir R1 (sekitar 3800 iterasi). Dengan demikian, dapat disimpulkan untuk *test case* kecil representasi terburuk adalah R1 dan yang terbaik adalah R4. Hasil ini juga memperkuat kesimpulan algoritma SPEA2 lebih baik dari AB.



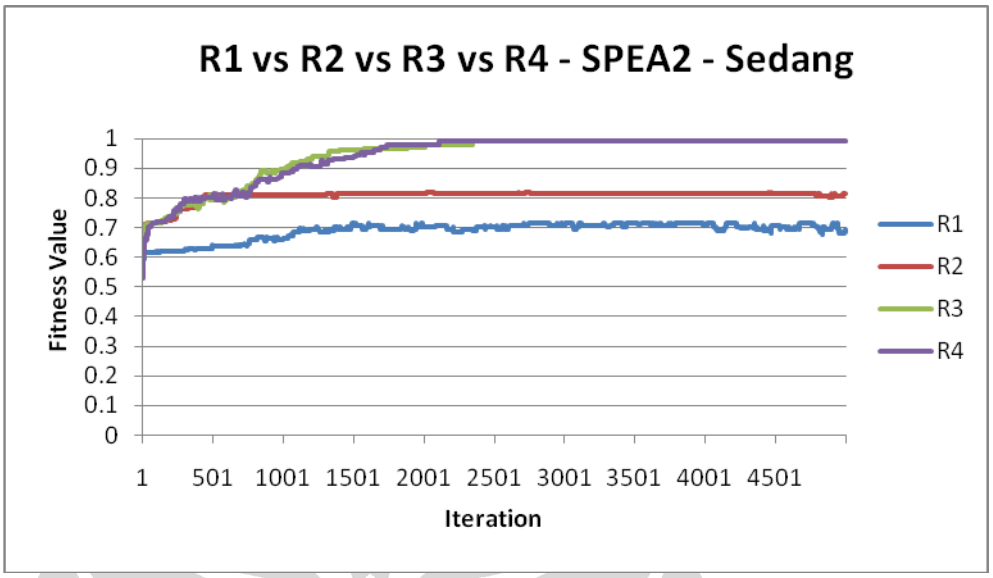
Gambar 5.6 R1 vs R2 vs R3 vs R4 - SPEA2 - Kecil



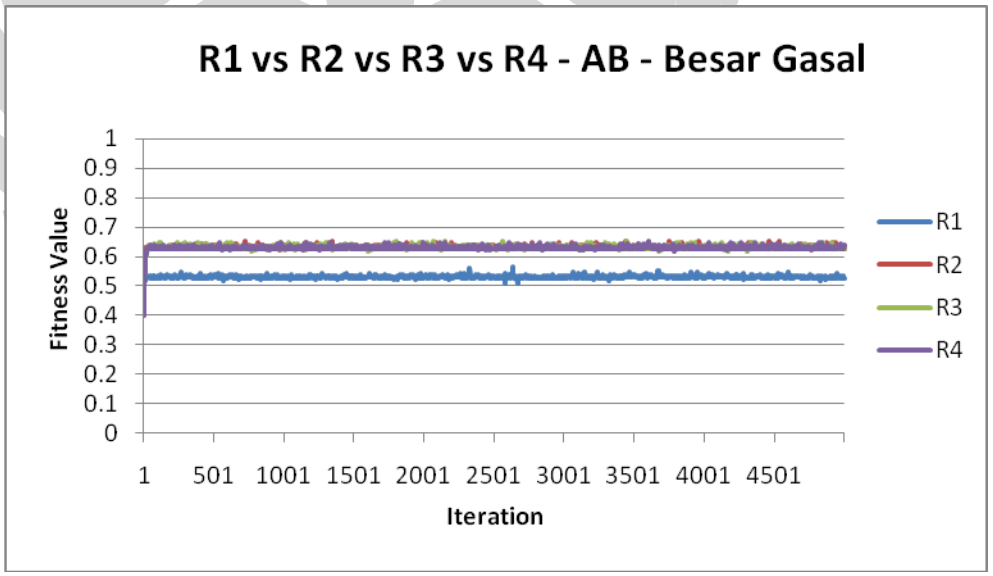
Gambar 5.7 R1 vs R2 vs R3 vs R4 - AB - Sedang

Dari Gambar 5.7 terlihat bahwa R1, R2, R3, dan R4 tidak dapat menemukan solusi optimal. R3 dan R4 hanya mencapai nilai *fitness* sekitar 0.8, R2 mencapai nilai *fitness* sekitar 0.72, dan R1 mencapai nilai *fitness* sekitar 0.62. Sementara, pada Gambar 5.8 terlihat R3 dan R4 masih bisa menemukan solusi optimal. R4 lebih cepat sedikit dari R3 dengan menemukan solusi optimal di sekitar 2200 iterasi, sementara R3 di sekitar 2400 iterasi. R2 dapat mencapai nilai *fitness* 0.82

dan R1 hanya dapat mencapai nilai *fitness* 0.73. Dengan demikian, dapat disimpulkan untuk *test case* sedang representasi terbaik adalah R4.



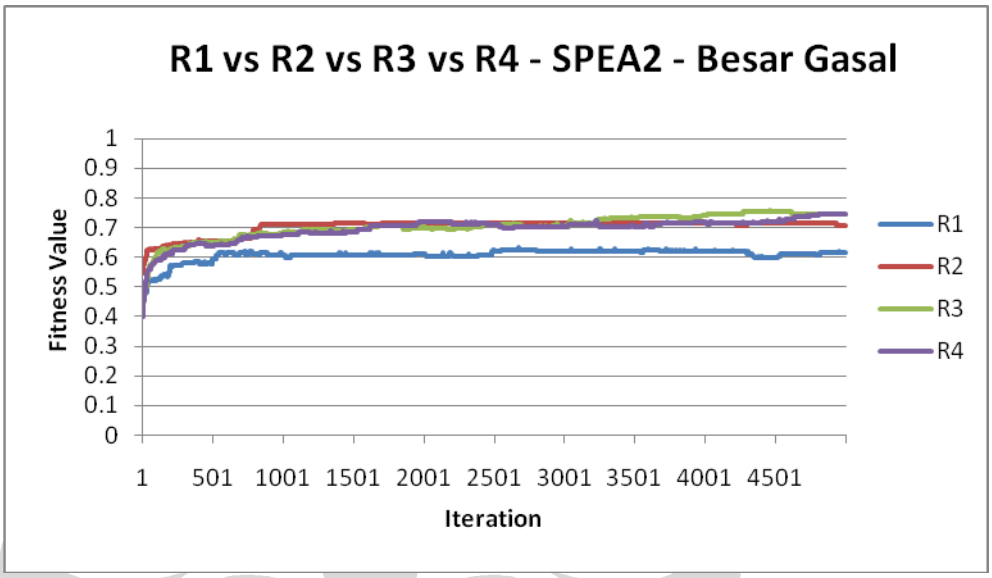
Gambar 5.8 R1 vs R2 vs R3 vs R4 - SPEA2 - Sedang



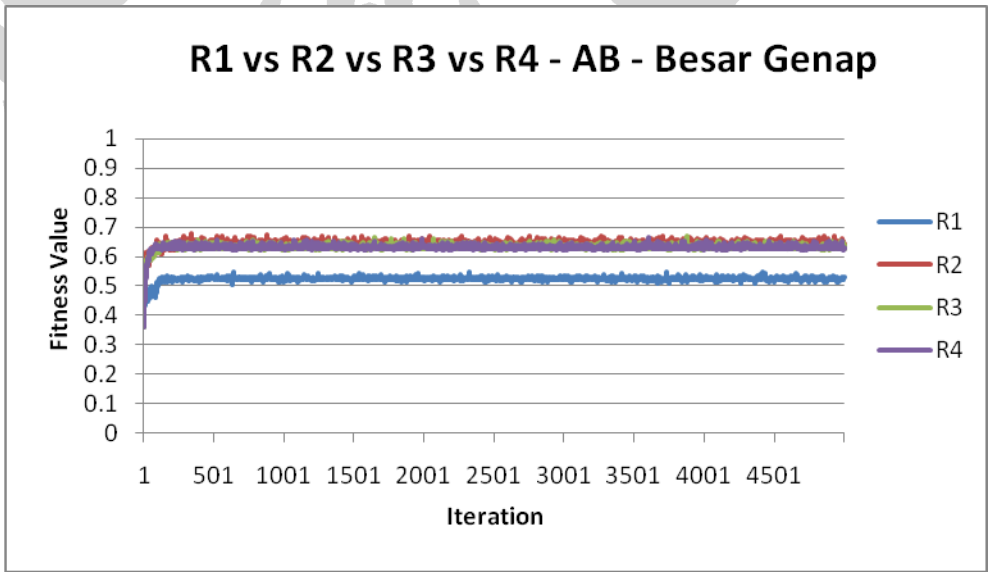
Gambar 5.9 R1 vs R2 vs R3 vs R4 - AB - Besar Gasal

Dari Gambar 5.9 terlihat bahwa R1, R2, R3, dan R4 tidak dapat menemukan solusi optimal. R2, R3, dan R4 hanya mencapai nilai *fitness* sekitar 0.65, sementara R1 hanya mencapai nilai *fitness* sekitar 0.57. Sementara, pada Gambar 5.10 terlihat tidak ada representasi yang dapat menemukan solusi optimal seperti di *test case* kecil dan sedang. R1 hanya dapat mencapai nilai *fitness* 0.62, R2 dapat

mencapai nilai *fitness* 0.71, R3 dapat mencapai nilai *fitness* 0.75 dan R4 dapat mencapai nilai *fitness* 0.74. Dengan demikian, dapat disimpulkan untuk *test case* besar gasal representasi terbaik adalah R3.



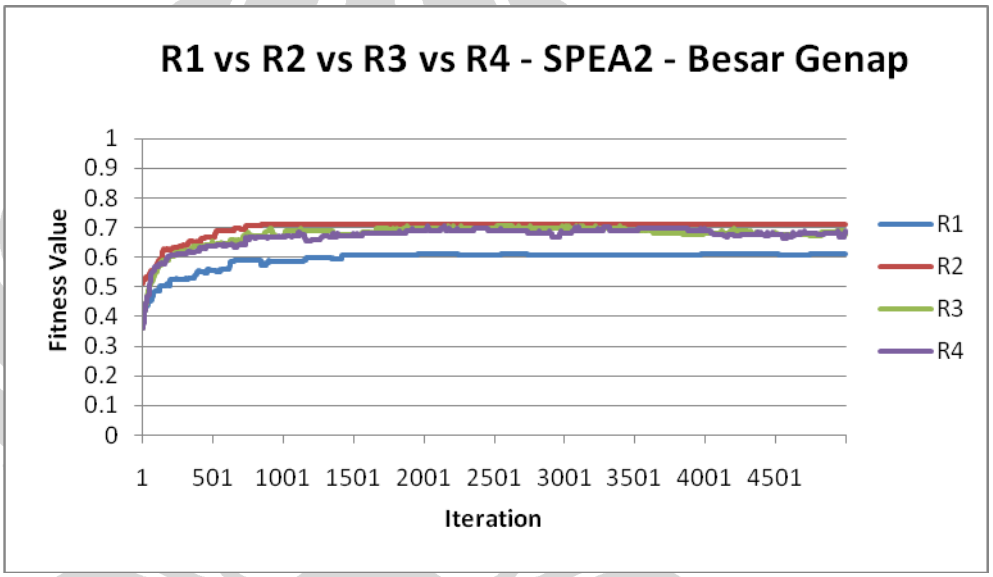
Gambar 5.10 R1 vs R2 vs R3 vs R4 - SPEA2 - Besar Gasal



Gambar 5.11 R1 vs R2 vs R3 vs R4 - AB - Besar Genap

Dari Gambar 5.11 terlihat bahwa R1, R2, R3, dan R4 tidak dapat menemukan solusi optimal karena memang sejak *test case* sedang, algoritma AB sudah tidak dapat menemukan solusi optimal. Pada Gambar 5.12 terlihat R1 hanya dapat

mencapai nilai *fitness* 0.6, sementara R2, R3, dan R4 dapat mencapai nilai *fitness* 0.68. Jika diperhatikan, R2 sedikit lebih unggul dari R3 dan R4. Hal ini dimungkinkan karena adanya perbedaan *state* yang dapat dicapai antara R2 dengan R3 dan R4. R2 memungkinkan adanya *state* dimana 1 MK diajar oleh lebih dari 1 dosen yang berbeda. Sementara R3 dan R4 menjamin bahwa 1 MK diajar oleh 1 dosen. Pada kenyataannya, memang tidak ada *constraint* yang mengatur hal tersebut, jadi keduanya sah-sah saja. Hal ini membuat R3 dan R4 lebih terikat dibandingkan dengan R2. Atau dengan kata lain, R2 memiliki *search space* yang lebih besar dari R3 dan R4 sehingga mempengaruhi nilai *fitness function* pada *test case* besar genap yang merupakan *test case* yang paling besar.

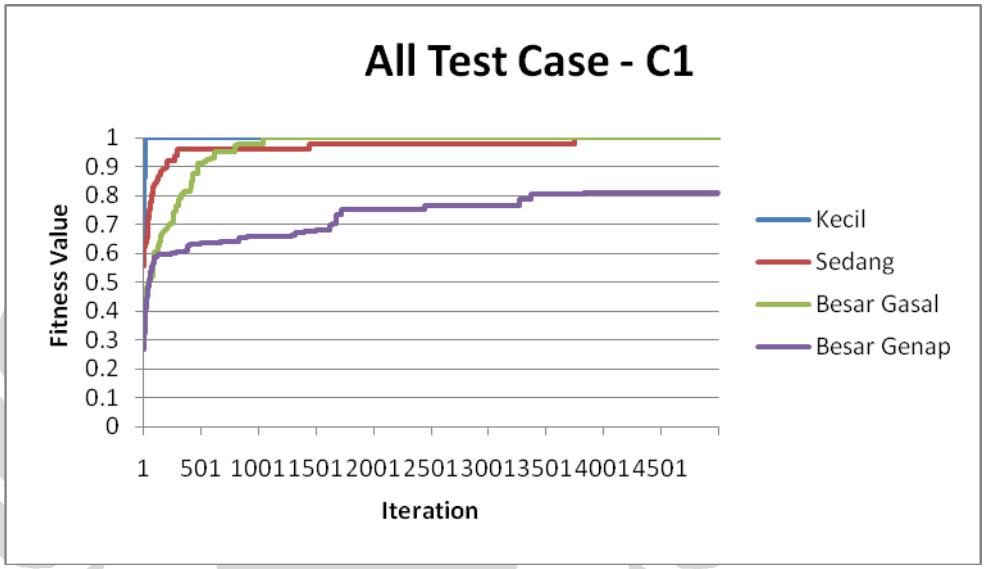


Gambar 5.12 R1 vs R2 vs R3 vs R4 - SPEA2 - Besar Genap

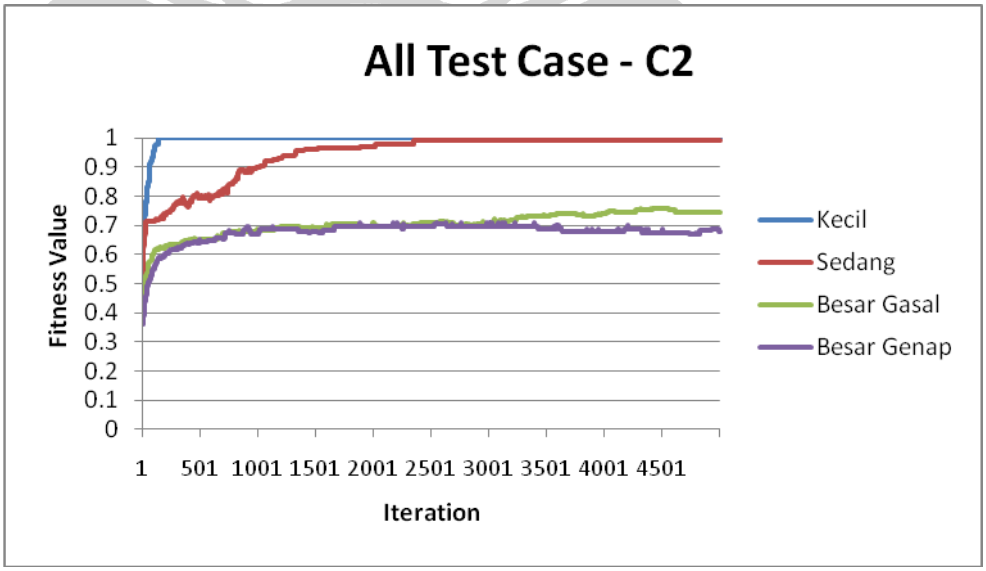
Kesimpulan dari hasil eksperimen ini yaitu secara umum hasil R3 dan R4 paling baik di antara representasi yang lain. *State* yang dapat dicapai R3 dapat pula dicapai R4, namun R3 memiliki representasi yang lebih fleksibel yaitu dapat menukar slot dengan mengubah nilai K dalam sekali mutasi, sementara R4 untuk melakukan hal yang sama membutuhkan beberapa kali mutasi. Oleh karena itu, ditarik kesimpulan bahwa representasi terbaik adalah R3 dan akan digunakan untuk eksperimen selanjutnya.

5.2 Hasil Performa GA Sebelum Parameter Dioptimasi

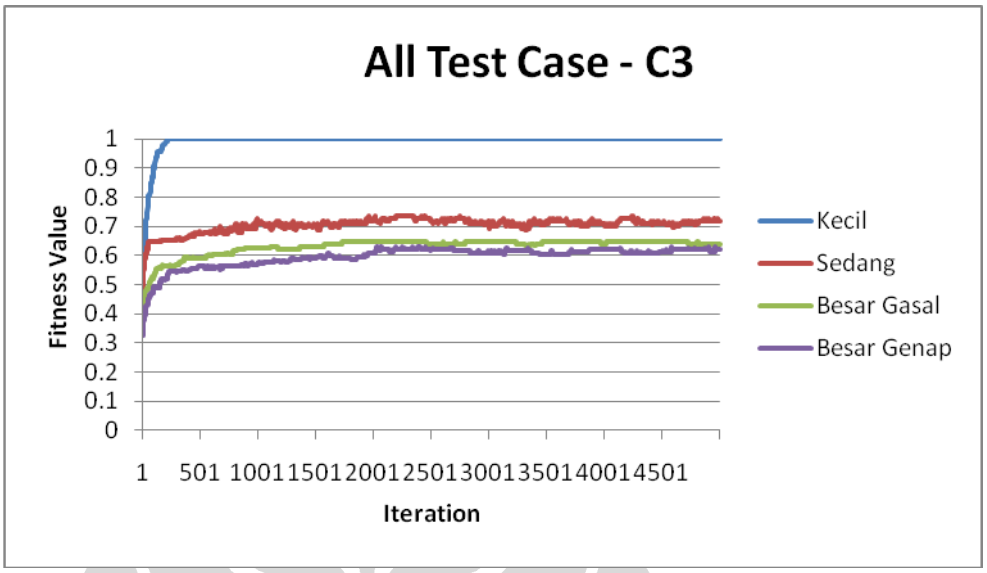
Dalam eksperimen performa GA sebelum parameter dioptimasi, yang dicoba adalah semua kombinasi dari ukuran *test case* (kecil, sedang, besar gasal, besar genap) dan *constraint* aktif (C1, C2, C3) dengan algoritma SPEA2, representasi R3, GA parameter: populasi 100, iterasi 5000, *crossover rate* 1, *mutation rate* 0.05, *crossover type* one, *archive size* 50%. Masing-masing dicoba dengan 5 *random seed* dan dirata-rata. Tujuan dari eksperimen ini adalah untuk mengetahui performa dari GA dengan parameter yang belum dioptimasi.



Gambar 5.13 *All Test Case* - C1



Gambar 5.14 *All Test Case* - C2



Gambar 5.15 *All Test Case* - C3

Gambar 5.13 - 5.15 menampilkan performa GA sebelum parameter dioptimasi pada setiap *test case* dipandang dari setiap *constraint* aktif yang ada. Kesimpulan-kesimpulan yang dapat diambil dari hasil eksperimen performa GA sebelum parameter dioptimasi adalah sebagai berikut.

- Dari Gambar 5.13 terlihat bahwa pada *constraint* aktif C1 setiap *test case* dapat menemukan solusi optimal kecuali *test case* besar genap. Kecepatan menemukan solusi optimal pada *test case* kecil sangat cepat yaitu hanya dalam 15 iterasi. Sementara kecepatan *test case* besar gasal lebih cepat dari *test case* sedang. Hal ini disebabkan karena ada satu eksperimen di *test case* sedang yang hasilnya buruk, sehingga ketika dirata-rata mempengaruhi hasil keseluruhan.
- Dari Gambar 5.14 terlihat bahwa pada *constraint* aktif C2, *test case* kecil dan sedang masih bisa menemukan solusi optimal, sedangkan *test case* besar gasal dan besar genap tidak menemukan solusi optimal.
- Dari Gambar 5.15 terlihat bahwa pada *constraint* aktif C3, hanya *test case* kecil yang berhasil menemukan solusi optimal. Tambahan satu *constraint* saja (SC6) membuat performa *test case* sedang turun dari yang tadinya berhasil menemukan solusi optimal menjadi hanya berhasil mencapai nilai *fitness*

sekitar 0.72. Hal ini karena SC6 memang dirancang untuk dioptimasi, bukan untuk ditemukan solusi optimal, sehingga sulit untuk tidak melanggar SC6.

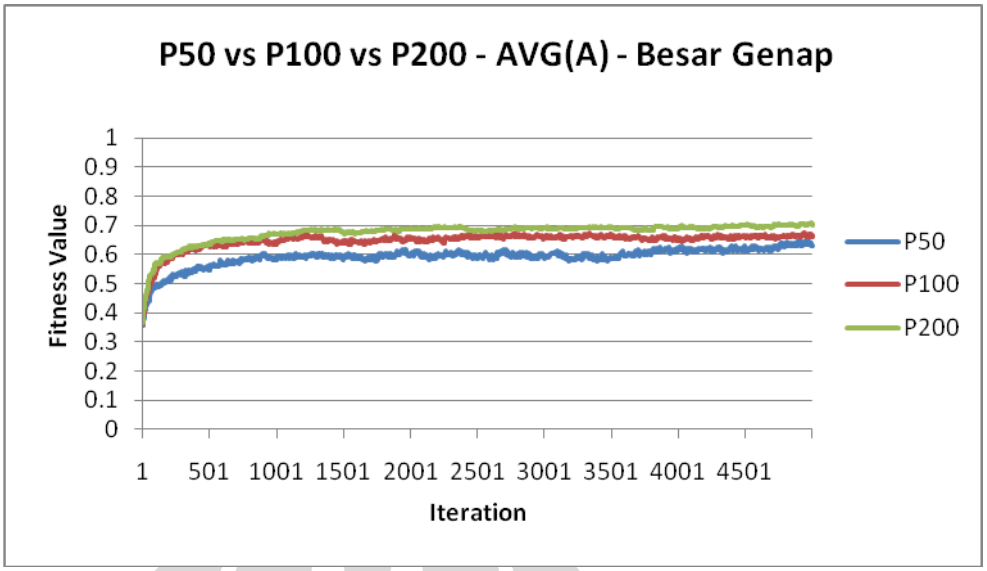
- Di semua *test case*, semakin banyak *constraint* aktif, semakin sulit ditemukan solusi optimal.

5.3 Hasil Penentuan Parameter Ukuran Populasi dan Archive Size

Dalam eksperimen penentuan parameter ukuran populasi dan *archive size*, dicoba semua kombinasi dari ukuran *test case* (kecil, sedang, besar gasal, besar genap), GA parameter populasi (P50, P100, P200), dan GA parameter *archive size* (A20, A50, A80) dengan algoritma SPEA2, representasi R3, GA parameter: iterasi 5000, *crossover rate* 1, *mutation rate* 0.05, *crossover type one*. Masing-masing dicoba dengan 5 *random seed* dan dirata-rata. Tujuan dari eksperimen ini adalah untuk mencari parameter populasi dan *archive size* terbaik.

5.3.1 Penentuan Parameter Ukuran Populasi

Dilihat dari Gambar 5.16, populasi 200 pada *test case* besar genap menghasilkan nilai *fitness* yang paling baik yaitu sekitar 0.7 jika dibandingkan dengan populasi 100 (sekitar 0.66) dan populasi 50 (sekitar 0.63). Hal ini karena semakin besar populasi, semakin besar kapasitas untuk menyimpan keragaman, semakin banyak individu yang dihasilkan per generasi, semakin besar kemungkinan menghasilkan solusi, dan semakin cepat menemukan solusi yang lebih baik dari generasi sebelumnya. Hasil eksperimen di *test case* lain juga menunjukkan tren yang sama yaitu semakin besar populasi, performa GA semakin baik.

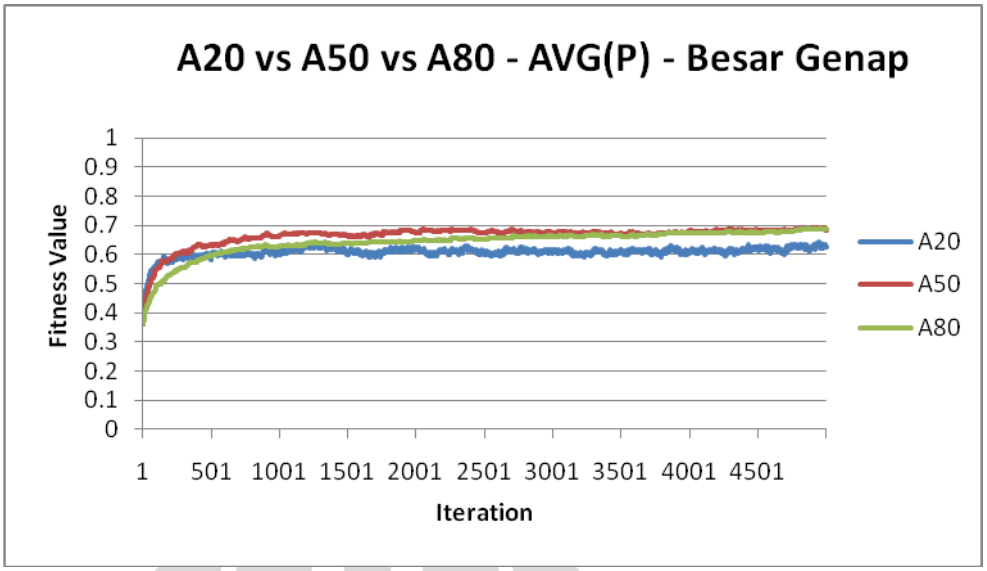


Gambar 5.16 P50 vs P100 vs P200 - AVG(A) - Besar Genap

5.3.1 Penentuan Parameter Ukuran Archive Size

Dilihat dari Gambar 5.17, *archive size* (lihat subbab 2.3.2) 20% dari jumlah populasi pada iterasi awal menang dari *archive size* 80%, namun pada iterasi akhir *archive size* 80% yang menang. Hal ini karena *archive* pada SPEA2 adalah tempat menyimpan individu terpilih (hasil *selection*), artinya generasi berikutnya ditentukan induk-induk yang ada di *archive*. Jika besar *archive* kecil, maka keragaman yang dapat ditampung lebih kecil dan akan terjadi *premature convergance* yaitu individu-individu dalam populasi akan cepat menjadi mirip dengan individu yang memiliki nilai *fitness* tinggi pada generasi tersebut.

Dapat dilihat juga *archive size* 50% pada *test case* besar gasal menghasilkan *fitness* yang paling baik yaitu sekitar 0.75 jika dibandingkan dengan *archive size* 80% (sekitar 0.71) dan *archive size* 20% (sekitar 0.68). Hal ini disebabkan karena 50% merupakan angka yang cocok untuk menghindari *premature convergance* dan juga tidak terlalu besar sehingga memperlambat konvergensi. Angka 50% ini ada juga erat kaitannya dengan masalah penjadwalan pada tugas akhir ini, dalam arti jika masalahnya lain, belum tentu *archive size* 50% menghasilkan *fitness* yang baik. Ada dugaan 50% merupakan jumlah yang pas untuk menyimpan individu *nondominated* yang penting untuk dilestarikan agar tidak mengalami *premature convergance*.

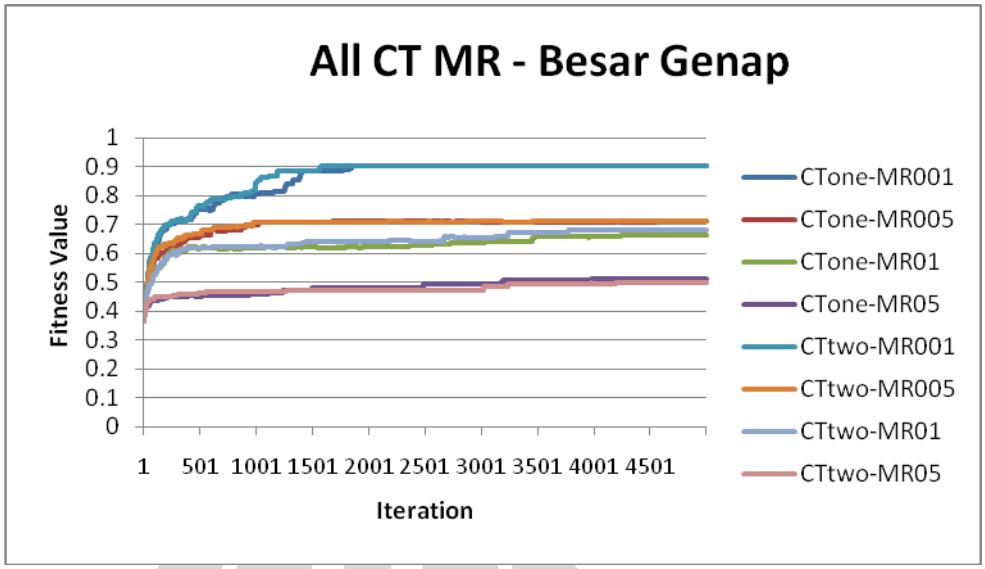


Gambar 5.17 A20 vs A50 vs A80 - AVG(P) - Besar Genap

Hasil eksperimen di *test case* lain juga menunjukkan tren yang sama yaitu *archive size* 20% pada iterasi awal menang dari *archive size* 80%, namun pada iterasi akhir *archive size* 80% yang menang dan *archive size* 50% menghasilkan *fitness* yang paling baik.

5.4 Hasil Penentuan Parameter Operator Genetik

Dalam eksperimen penentuan parameter operator, dicoba semua kombinasi dari ukuran *test case* (kecil, sedang, besar gasal, besar genap), GA parameter *mutation rate* (MR05, MR01, MR005, MR001), GA parameter *crossover rate* (CR05, CR1), dan GA parameter *crossover type* (CTone, CTtwo) (penjelasan *crossover type* lihat subbab 3.5) dengan algoritma SPEA2, representasi R3, GA parameter: populasi 200, iterasi 5000, *archive size* 50%. Masing-masing dicoba dengan 5 *random seed* dan dirata-rata. Tujuan dari eksperimen ini adalah untuk mencari *crossover rate* terbaik, *crossover type* terbaik, dan *mutation rate* terbaik.



Gambar 5.18 *All* CT MR - Besar Genap

Gambar 5.18 menggambarkan performa semua kombinasi *crossover type* dan *mutation rate* pada *test case* besar genap. Parameter *crossover rate* tidak ditampilkan karena ternyata tidak berpengaruh pada eksperimen karena pada ECJ, *crossover rate* hanya diperhitungkan pada *crossover type any*, tidak diperhitungkan pada *crossover type one* dan *two*. Kesimpulan-kesimpulan yang dapat diambil dari eksperimen penentuan parameter operator genetik adalah sebagai berikut.

- Pada Gambar 5.18 terlihat nilai *fitness mutation rate* yang sama dengan *crossover type* berbeda hasilnya tidak terlalu jauh. Hal ini menunjukkan bahwa *mutation rate* lebih berpengaruh terhadap performa GA daripada *crossover type*.
- Pada Gambar 5.18 terlihat untuk *crossover type two*, *mutation rate* 0.5 menghasilkan nilai *fitness* sekitar 0.49, *mutation rate* 0.1 menghasilkan nilai *fitness* sekitar 0.68, *mutation rate* 0.05 menghasilkan nilai *fitness* sekitar 0.72, dan *mutation rate* 0.01 menghasilkan nilai *fitness* sekitar 0.9. Dapat disimpulkan *mutation rate* yang lebih kecil menghasikan nilai *fitness* yang lebih baik dan *mutation rate* terbaik adalah 0.01. Hal ini disebabkan karena panjang *genome* dari *test case* besar genap adalah 354, jika *mutation rate* 0.05 maka jumlah rata-rata *genome* yang dimutasi adalah 17.7. Ada mutasi yang membuat *fitness* value lebih baik, ada pula yang membuat *fitness* value

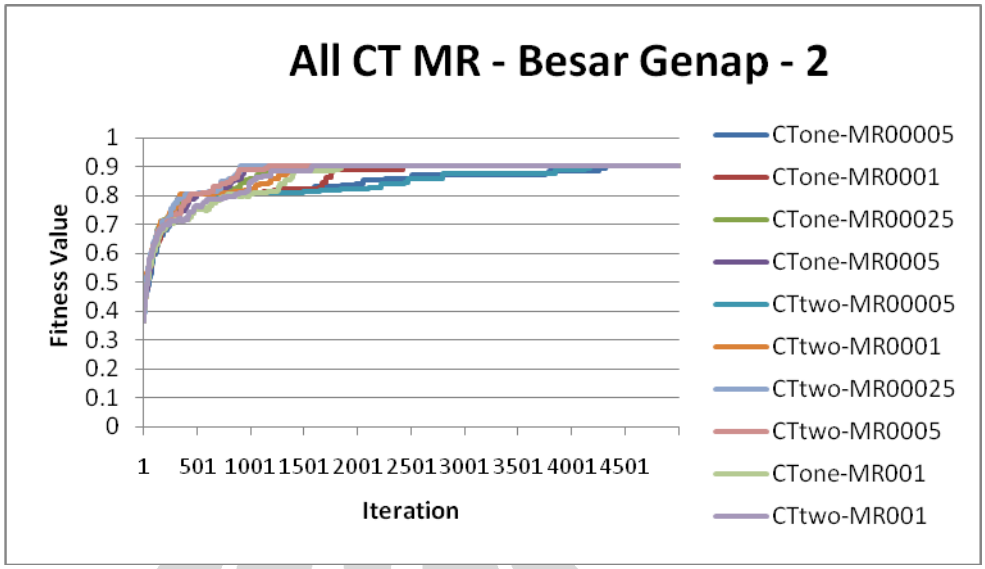
menjadi buruk, ada pula yang netral. Ketika terlalu banyak *genome* yang dimutasi, misalnya 17, besar kemungkinan mutasi baik ditiadakan oleh mutasi buruk. Lebih sulit mencari 17 mutasi baik sekaligus dibandingkan dengan 1 sekaligus. Jika *mutation rate* 0.01 maka jumlah rata-rata *genome* yang dimutasi adalah 3.54, oleh karena itu lebih besar kemungkinannya terjadi mutasi baik tanpa ditiadakan mutasi buruk.

- Secara umum perbedaan hasil *crossover type one* dan *crossover type two* tidak terlalu signifikan. Ada kasus di mana *type one* lebih unggul, ada juga kasus di mana *type two* lebih unggul. Karena *crossover type two* lebih unggul di *mutation rate* terbaik dan di *test case* paling besar yaitu besar genap, maka diambil kesimpulan *crossover type two* adalah yang terbaik dan akan digunakan pada eksperimen selanjutnya.

5.4.1 Eksperimen Penentuan Parameter Operator Genetik Kedua

Eksperimen ini dilakukan karena hasil eksperimen penentuan parameter operator genetik menunjukkan bahwa *mutation rate* 0.01 paling baik, sementara 0.01 adalah *mutation rate* terkecil yang dicobakan. Muncul pertanyaan apakah kalau *mutation rate* dibuat lebih kecil lagi akan menghasilkan *fitness function* yang lebih baik atau lebih cepat konvergen. Oleh karena itu, diadakan eksperimen kembali untuk *mutation rate* 0.005, 0.0025, 0.001, 0.0005 hanya pada *test case* besar genap. Pengambilan keputusan eksperimen hanya pada *test case* besar genap karena hanya *test case* ini yang belum menemukan solusi optimal dan karena keterbatasan waktu.

Dalam eksperimen penentuan parameter operator genetik kedua, dicoba kombinasi dari GA parameter *mutation rate* (MR0005, MR00025, MR0001, MR00005) dan GA parameter *crossover type* (CTone, CTtwo) dengan ukuran *test case* besar genap, algoritma SPEA2, representasi R3, GA parameter: populasi 200, iterasi 5000, *archive size* 50%. Masing-masing dicoba dengan 5 *random seed* dan dirata-rata. Tujuan dari eksperimen ini adalah untuk mencari *crossover type* terbaik dan *mutation rate* terbaik.

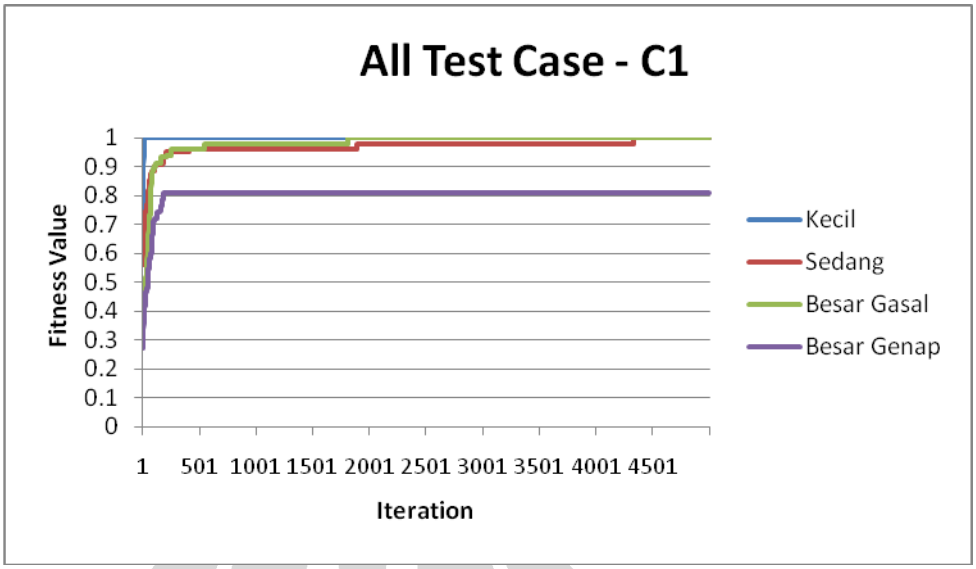


Gambar 5.19 *All* CT MR - Besar Genap - 2

Dilihat dari Gambar 5.19, *crossover type two* dengan *mutation rate* 0.0025 adalah yang paling cepat mencapai nilai *fitness* 0.9 yaitu pada sekitar iterasi 1000. Jadi, kesimpulan dari eksperimen ini adalah ternyata benar 0.01 belum merupakan *mutation rate* terbaik, yang terbaik adalah *mutation rate* 0.0025 dengan *crossover type two*. Penjelasannya sama dengan eksperimen 4, yaitu jika *mutation rate* 0.0025 maka jumlah rata-rata *genome* yang dimutasi adalah 0.885. Oleh karena itu, lebih besar kemungkinannya terjadi mutasi baik tanpa ditidakan mutasi buruk. Namun, jika *mutation rate* diperkecil lagi, akan membuat *mutation* jarang terjadi sehingga memperlambat mendapatkan individu yang ideal.

5.5 Hasil Performa GA Sesudah Parameter Dioptimasi

Dalam eksperimen performa GA sesudah parameter dioptimasi, dicoba semua kombinasi dari ukuran *test case* (kecil, sedang, besar gasal, besar genap) dan *constraint* aktif (C1, C2, C3) dengan algoritma SPEA2, representasi R3, dan GA parameter: populasi 200, *archive size* 50%, iterasi 5000, *mutation rate* 0.0025, *crossover type two*. Masing-masing dicoba dengan 5 *random seed* dan dirata-rata. Tujuan dari eksperimen ini adalah untuk mengetahui performa dari GA dengan parameter yang sudah dioptimasi.



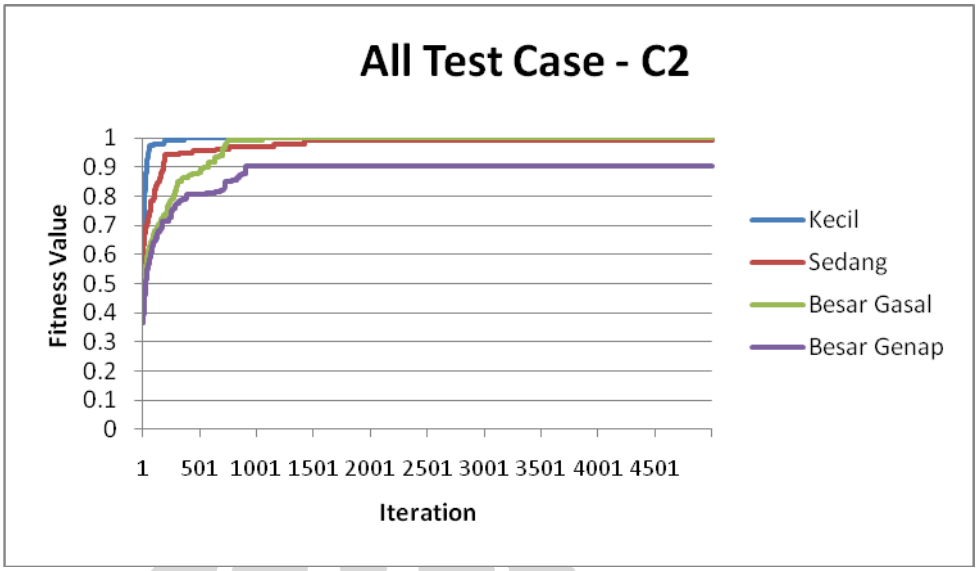
Gambar 5.20 *All Test Case* - C1

Gambar 5.20 - 5.22 menampilkan performa GA sesudah parameter dioptimasi pada setiap *test case* dipandang dari setiap *constraint* aktif yang ada. Kesimpulan-kesimpulan yang dapat diambil dari hasil eksperimen performa GA sesudah parameter dioptimasi adalah sebagai berikut.

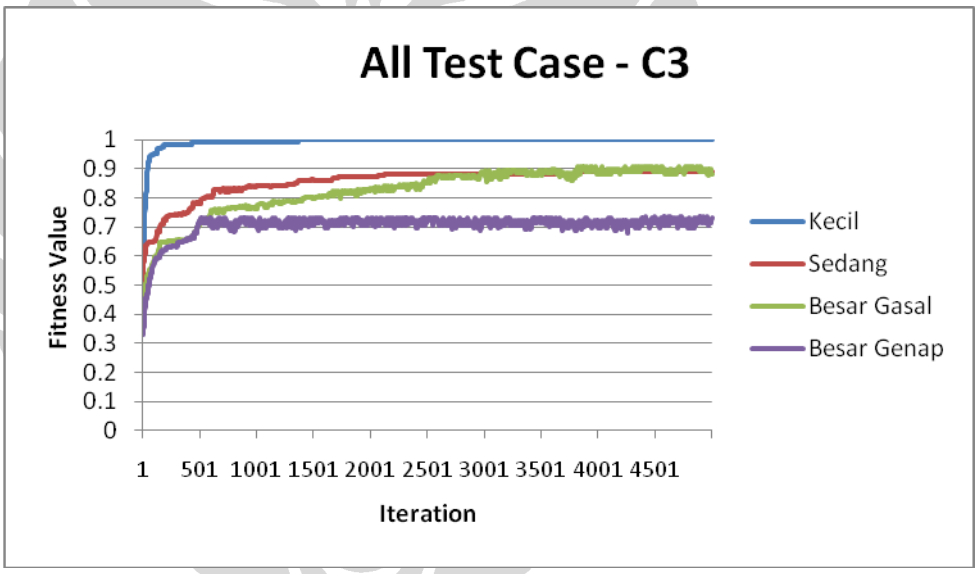
- Pada Gambar 5.20, dapat dilihat semua *test case* kecuali *test case* besar genap berhasil ditemukan solusi optimal dalam kurang dari 4500 iterasi. Setelah dilihat, *test case* besar genap selalu gagal dengan hasil terbaiknya yaitu pelanggaran *constraint* di HC 2 sebanyak 24 pelanggaran. Ternyata memang *test case* besar genap tidak bisa ditemukan solusi optimal dengan alasan ada 39 slot yang tersedia, sementara total sks MK wajib di tingkat 1 ada 48. Berarti agar MK wajib tingkat 1 tidak bentrok dibutuhkan minimal 48 slot. Padahal, hanya tersedia 39 slot untuk diisi. Hal ini akan menyebabkan pelanggaran di HC2 minimal sejumlah 48-39=9, yang mengindikasikan ada 9 MK wajib setingkat (tingkat 1) yang bentrok. Lebih lanjut lagi, yang tadi baru MK tingkat 1, masih ada tingkat 2,3, dan 4. Total sks MK wajib di tingkat masing-masing yaitu: tingkat 2 ada 54, tingkat 3 ada 30, dan tingkat 4 ada 4. Berarti tingkat 2 juga melebihi total slot yang ada, sementara tingkat 3 dan 4 tidak melebihi. Total pelanggaran di HC2 yang disebabkan MK tingkat 2 adalah 54-39=15, yang mengindikasikan ada 15 MK wajib setingkat (tingkat 2) yang bentrok. Berarti, jika pelanggaran di tingkat 1 dan tingkat 2 ditotal,

ada 9+15=24 pelanggaran. Ya, 24 pelanggaran, sama dengan hasil terbaik yang dapat dihasilkan GA pada *test case* besar genap. Dapat disimpulkan bahwa GA memang sudah maksimal, dan seharusnya bisa menemukan solusi optimal.

- Pada Gambar 5.21, dapat dilihat bahwa untuk *test case* kecil, sedang, dan besar gasal berhasil menemukan solusi optimal dalam kurang dari 1500 iterasi. Sedangkan *test case* besar genap, jika pelanggaran HC2 sebanyak 24 dianggap sudah optimal, berhasil ditemukan solusi optimal dalam kurang dari 1000 iterasi.
- Terjadi suatu keanehan pada hasil di C1 dan C2 dimana *test case* yang besar genap dapat ditemukan solusi optimal lebih cepat. Hal ini terjadi karena memang eksperimen sebelumnya, khususnya eksperimen penentuan parameter operator genetik kedua (lihat subbab 5.4.1) mencoba melakukan optimasi parameter untuk *test case* besar genap. Ternyata, akibatnya mempengaruhi kecepatan konvergensi di *test case* lain. Penurunan kecepatan konvergensi di *test case* selain besar genap disebabkan oleh *mutation rate* yang terlalu kecil yaitu 0.0025. Hal ini memperlambat konvergensi karena muation jarang terjadi seperti yang dijelaskan pada subbab 5.4.1. Beda halnya dengan *test case* besar genap, *mutation rate* 0.0025 adalah angka yang ideal. Jadi penentuan *mutation rate* yang ideal sangat tergantung pada masalahnya (panjang *genome*).
- Pada Gambar 5.22, dapat dilihat bahwa hanya *test case* kecil yang berhasil menemukan solusi optimal. Hal ini karena SC6 memang dirancang untuk dioptimasi, bukan untuk ditemukan solusi optimal. *Test case* kecil bisa menemukan solusi optimal karena memang semua mata kuliahnya dirancang wajib dan 1 tingkat, di mana jika terjadi bentrok tidak terkena pinalti.



Gambar 5.21 *All Test Case* - C2



Gambar 5.22 *All Test Case* - C3

Jika hasil eksperimen performa GA sesudah parameter dioptimasi dibandingkan dengan hasil eksperimen performa GA sebelum parameter dioptimasi (subbab 5.2), kesimpulan-kesimpulan yang dapat diambil adalah sebagai berikut.

- Untuk *constraint* aktif C1, performa GA sesudah parameter dioptimasi pada *test case* besar genap lebih cepat konvergen, sementara *test case* sedang dan besar gasal lebih lambat. Pada gambar 5.20, dapat dilihat *test case* besar genap sudah mencapai nilai *fitness* 0.8 pada sekitar iterasi ke

200, *test case* sedang mencapai solusi optimal pada sekitar iterasi ke 4400, dan *test case* besar gasal mencapai solusi optimal pada sekitar iterasi ke 1800. Sementara, pada gambar 5.13, dapat dilihat *test case* besar genap baru mencapai nilai *fitness* 0.8 pada sekitar iterasi ke 3300, *test case* sedang mencapai solusi optimal pada sekitar iterasi ke 3800, dan *test case* besar gasal mencapai solusi optimal pada sekitar iterasi ke 1200.

- Untuk *constraint* aktif C2, performa GA sesudah parameter dioptimasi lebih baik. Hal ini ditunjukkan oleh ditemukannya solusi optimal pada *test case* besar gasal dan besar genap (jika pelanggaran HC2 sebanyak 24 sudah dianggap optimal) yang dapat dilihat pada Gambar 5.21, padahal sebelum parameter dioptimasi, solusi optimal tidak ditemukan pada *test case* besar gasal dan besar genap (lihat gambar 5.14).
- Untuk *constraint* aktif C3, performa GA sesudah parameter dioptimasi juga lebih baik. Hal ini ditunjukkan oleh peningkatan pencapaian nilai *fitness* di *test case* sedang, besar gasal, dan besar genap. Untuk gambar performa GA sesudah parameter dioptimasi dengan *constraint* aktif C3 dapat dilihat pada Gambar 5.22, sedangkan untuk gambar performa GA sebelum parameter dioptimasi dengan *constraint* aktif C3 dapat dilihat pada Gambar 5.15. Dari kedua gambar tersebut dapat dilihat peningkatan di *test case* sedang yaitu dari nilai *fitness* 0.72 ke 0.9, peningkatan di *test case* besar gasal yaitu dari nilai *fitness* 0.65 ke 0.9, dan peningkatan di *test case* besar genap yaitu dari nilai *fitness* 0.61 ke 0.72.