PENERAPAN ALGORITMA GENETIKA PADA MASALAH PENJADWALAN OPERASI SISTEM PEMBANGKIT TENAGA LISTRIK

SKRIPSI

Oleh:

Faiqotul Himmah
NIM 051810101064

JURUSAN MATEMATIKA
FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM
UNIVERSITAS JEMBER
2013

PENERAPAN ALGORITMA GENETIKA PADA MASALAH PENJADWALAN OPERASI SISTEM PEMBANGKIT TENAGA LISTRIK

SKRIPSI

diajukan guna melengkapi tugas akhir dan memenuhi salah satu syarat
untuk menyelesaikan Program Studi Matematika (S1)
dan mencapai gelar Sarjana Sains

Oleh:

Faiqotul Himmah
NIM 051810101064

JURUSAN MATEMATIKA
FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM
UNIVERSITAS JEMBER
2013

PERSEMBAHAN

Skripsi ini saya persembahkan untuk:

1. Ibunda Supiyati A, Ma dan ayahanda Tjipto Hadi Suparto, S.Sos yang telah membesarkan, mendidik dan mendo'akan dengan penuh cinta & perhatian yang tak pernah putus;
2. Nenek Artijah, Hapipa, Suginten, dan Budhe Hosniyah Syafi'un yang dengan sabar telah mendidik dan menjadi guru kehidupan;
3. kakak-kakak tercinta (Mas Didik, Mas Tulus, Mbak Anik, Mbak Lia) dan keponakan tersayang (Deni dan Abdi) atas dukungan moril dan materiil;
4. keluarga besar Madura, Mas Ok dan Mbak Yuyun, Ca' Pi'i dan Mbak Mahwiyah serta keluarga besar Malang, Mas Hadi Suyono dan Mbak Peni;
5. guru-guru sejak TK hingga Perguruan Tinggi yang telah memberi ilmu;
6. Almamater Jurusan Matematika Fakultas MIPA Universitas Jember.

MOTO

Allah akan meninggikan orang-orang yang beriman di antara kamu dan orang-orang yang diberi ilmu pengetahuan beberapa derajat.

(terjemahan Surat *Al-Mujadalah* ayat 11)[*)]

*) Departemen Agama Republik Indonesia. 2005. *Al Qur'an dan Terjemahnya*. Bandung: PT Syaamil Cipta Media

PERNYATAAN

Saya yang bertanda tangan di bawah ini:

nama : Faiqotul Himmah

NIM : 051810101064

menyatakan dengan sesungguhnya bahwa karya ilmiah yang berjudul ”Penerapan Algoritma Genetika pada Masalah Penjadwalan Operasi Sistem Pembangkit Tenaga Listrik” adalah benar-benar hasil karya sendiri, kecuali kutipan yang telah saya sebutkan sumbernya, belum pernah diajukan pada institusi mana pun, dan bukan karya jiplakan. Saya bertanggung jawab atas keabsahan dan kebenaran isinya sesuai dengan sikap ilmiah yang harus dijunjung tinggi.

Demikian pernyataan ini saya buat dengan sebenarnya, tanpa ada tekanan dan paksaan dari pihak mana pun serta bersedia mendapat sanksi akademik jika ternyata di kemudian hari pernyataan ini tidak benar.

Jember, Januari 2013

Yang menyatakan,

Faiqotul Himmah
NIM 051810101064

SKRIPSI

PENERAPAN ALGORITMA GENETIKA PADA MASALAH PENJADWALAN OPERASI SISTEM PEMBANGKIT TENAGA LISTRIK

Oleh

Faiqotul Himmah
NIM 051810101064

Pembimbing

Dosen Pembimbing Utama	:	Agustina Pradjaningsih, S.Si, M.Si
Dosen Pembimbing Anggota	:	Kiswara Agung Santoso, S.Si, M.Kom

PENGESAHAN

Skripsi berjudul " Penerapan Algoritma Genetika pada Masalah Penjadwalan Operasi Sistem Pembangkit Tenaga Listrik" telah diuji dan disahkan pada:

hari, tanggal :

tempat : Fakultas Matematika dan Ilmu Pengetahuan Alam Universitas Jember

Tim Penguji:

Ketua,	Sekretaris,
Agustina Pradjaningsih, S.Si, M.Si NIP 197108022000032009	Kiswara Agung Santoso, S.Si, M.Kom NIP 197209071998031003
Penguji I,	Penguji II,
Drs. Moh. Hasan, M.Sc, Ph.D NIP 196404041988021001	Bagus Juliyanto, SSi NIP 198007022003121001

Mengesahkan
Dekan,

Prof. Drs. Kusno, DEA, Ph.D.
NIP 196101081986021001

RINGKASAN

Penerapan Algoritma Genetika pada Masalah Penjadwalan Operasi Sistem Pembangkit Tenaga Listrik ; Faiqotul Himmah, 051810101064; 2012: 38 halaman; Jurusan Matematika Fakultas Matematika dan Ilmu Pengetahuan Alam Universitas Jember.

Kehidupan manusia saat ini tidak dapat dipisahkan dari listrik. Kebutuhan listrik tersebut disuplai oleh unit-unit pembangkit listrik yang saling terhubung membentuk sistem tenaga listrik. Biaya operasi sistem tenaga listrik merupakan bagian biaya terbesar bagi suatu perusahaan listrik dan 60% dari biaya tersebut adalah biaya bahan bakar. Listrik yang dibangkitkan oleh sistem tenaga listrik sangat tergantung pada kebutuhan pelanggan yang selalu mengalami perubahan dari waktu ke waktu. Oleh karena itu harus dilakukan penjadwalan operasi sistem tenaga listrik agar biaya pembangkitan minimal. Masalah penjadwalan operasi sistem tenaga listrik ini disebut dengan *unit commitment*. *Unit commitment* berkaitan erat dengan masalah *economic dispatch* yakni masalah menentukan berapa besar daya yang dibangkitkan oleh unit yang beroperasi. Penelitian bertujuan mendapatkan jadwal operasi sistem pembangkit tenaga listrik yang optimal dengan menerapkan *fitness scaling* dalam algoritma genetika.

Algoritma genetika bekerja dengan membangkitkan sekumpulan jadwal pembangkitan secara acak. Kemudian dihitung biaya pembangkitan dari masing-masing jadwal tersebut. Biaya pembangkitan ini kemudian ditransformasi menjadi nilai *fitness*. Untuk meningkatkan performa algoritma genetika, dilakukan *fitness scaling*. Jenis *fitness scaling* yang digunakan adalah *exponential scaling*. Selanjutnya jadwal pembangkitan direproduksi melalui tahapan seleksi, *crossover*, dan mutasi sehingga diperoleh jadwal pembangkitan yang baru dengan biaya pembangkitan lebih minimal dari jadwal sebelumnya.

Data yang digunakan dalam penelitian ini adalah data karakteristik pembangkit pada PLN PJB II yang beroperasi di Jawa Timur dan Bali. Terdapat lima belas unit pembangkit yang siap beroperasi dengan beban harian selama 24 jam. Ukuran populasi yang digunakan adalah 30 yakni terdapat 30 jadwal pembangkitan dalam satu iterasi dengan probabilitas *crossover* (p_c) sebesar 0,85, probabilitas mutasi (p_m) sebesar 0,00009, dan iterasi maksimum sebanyak 1000. Jadwal pembangkitan terbaik diperoleh pada iterasi ke-920 dengan biaya pembangkitan sebesar Rp 17.093.816.034 (tujuh belas milyar sembilan puluh tiga juta delapan ratus enam belas ribu tiga puluh empat rupiah).

PRAKATA

Segala puji bagi Allah SWT, dengan rahmat dan pertolongan-Nya penulis dapat menyelesaikan skripsi berjudul **Penerapan Algoritma Genetika pada Masalah Penjadwalan Operasi Sistem Pembangkit Tenaga Listrik**. Pada kesempatan ini penulis menyampaikan ucapan terima kasih kepada:

1. Agustina Pradjaningsih, S.Si, M.Si selaku Dosen Pembimbing Utama, Kiswara Agung Santoso, M.Kom selaku Dosen Pembimbing Anggota, dan Drs. Moh. Hasan, M.Sc, Ph.D selaku Dosen Pembimbing Akademik sekaligus Dosen Penguji dan Bagus Julianto, S.Si selaku Dosen Penguji atas waktu, perhatian dan bimbingannya selama studi serta dalam penyusunan skripsi;
2. seluruh Dosen dan karyawan di Fakultas MIPA khususnya Jurusan Matematika;
3. Lailin Nadzifah, S.Pd, Laily Chusnul Chotimah, S.E, dan seluruh sahabat perjuangan yang telah memberi dukungan dan do'a tiada henti untuk terselesaikannya skripsi ini;
4. Eka, Anik, Yulia, Titi, Valentia, dan seluruh mahasiswa Jurusan Matematika yang telah memberi semangat, bantuan dan inspirasi;
5. Arin, Dewi, Reni, Gita, dan alumni SMASA Pamekasan yang telah banyak membantu;
6. semua pihak yang tidak dapat disebutkan satu per satu.

Semoga karya ilmiah ini menjadi salah satu pustaka yang bermanfaat bagi pembaca.

Jember, Januari 2013 Penulis

DAFTAR ISI

DAFTAR TABEL

Halaman

DAFTAR GAMBAR

Halaman

DAFTAR LAMPIRAN

Halaman

BAB 1. PENDAHULUAN

1.1 Latar Belakang

Listrik merupakan salah satu sumber energi terpenting dalam kehidupan modern. Listrik saat ini sudah menjadi kebutuhan pokok bagi masyarakat dan industri. Hampir semua peralatan yang dekat dengan kehidupan manusia selalu membutuhkan listrik.

Listrik yang dikonsumsi masyarakat disuplai dari berbagai unit pembangkit listrik yang saling terhubung. Unit-unit pembangkit yang terhubung ini membentuk kesatuan interkoneksi yang kemudian disebut dengan sistem tenaga listrik. Operasi sistem tenaga listrik tentu membutuhkan biaya. Biaya operasi sistem tenaga listrik (diantaranya adalah biaya pembelian tenaga listrik, biaya pegawai, biaya bahan bakar dan material operasi) merupakan bagian biaya terbesar bagi suatu perusahaan listrik. Sementara itu, biaya bahan bakar menghabiskan 60 persen dari biaya operasi secara keseluruhan (Marsudi, 2006: 6).

Sementara itu beban sistem sangat tergantung pada kebutuhan tenaga listrik pelanggan yang selalu mengalami perubahan dari waktu ke waktu. Dalam satu hari misalnya, total beban akan tinggi pada sore hari karena banyak industri beroperasi dan banyak lampu dihidupkan sedangkan pada tengah malam atau pada pagi hari total kebutuhan beban rendah. Demikian juga dalam seminggu, pada hari Sabtu atau Minggu kebutuhan beban lebih rendah daripada hari-hari sebelumnya karena pada hari Sabtu atau Minggu kantor-kantor dan industri tidak beroperasi.

Dengan biaya operasional yang besar dan beban yang selalu berubah dari waktu ke waktu tentu sangat tidak efisien jika mengoperasikan terlalu banyak unit pembangkit (pusat listrik) pada saat kebutuhan beban rendah. Oleh karena itu, penjadwalan operasi sistem pembangkit tenaga listrik menjadi masalah penting. Penjadwalan operasi sistem pembangkit tenaga listrik didasarkan pada prinsip meminimalkan biaya bahan bakar dengan kendala beban sistem yang ada. Masalah ini disebut dengan *unit commitment* pada pembangkit listrik. Wibowo (2003) telah

menyelesaikan masalah UC dengan menggunakan metode pemrograman dinamis *fuzzy*. Sementara, untuk masalah optimasi, banyak algoritma yang bisa digunakan. Termasuk algoritma yang menggunakan metode probabilistik. Salah satu yang populer adalah algoritma genetika.

Dalam skripsi ini penulis akan menerapkan algoritma genetika dengan menerapkan *fitness scaling*. Algoritma genetika dikenal sebagai algoritma yang mampu menyelesaikan permasalahan yang kompleks yang sukar diselesaikan dengan metode konvensional. Sementara itu, penambahan proses *fitness scaling* diharapkan mampu memberikan solusi optimal karena kemampuannya untuk mempertahankan kromosom terbaik.

1.2 Perumusan Masalah

Masalah yang akan dibahas dalam skripsi ini adalah bagaimana mendapatkan jadwal operasi sistem pembangkit tenaga listrik optimal dengan beberapa asumsi sebagai berikut:

a. periode penjadwalan adalah periode jangka pendek, tiap periode dibagi atas beberapa interval (selang waktu) yang sama panjang yakni satu jam;
b. besar beban pada periode tertentu dianggap diketahui dengan pasti.

1.3 Tujuan

Skripsi ini bertujuan untuk mendapatkan jadwal operasi sistem pembangkit tenaga listrik yang optimal dengan menerapkan *fitness scaling* dalam algoritma genetika.

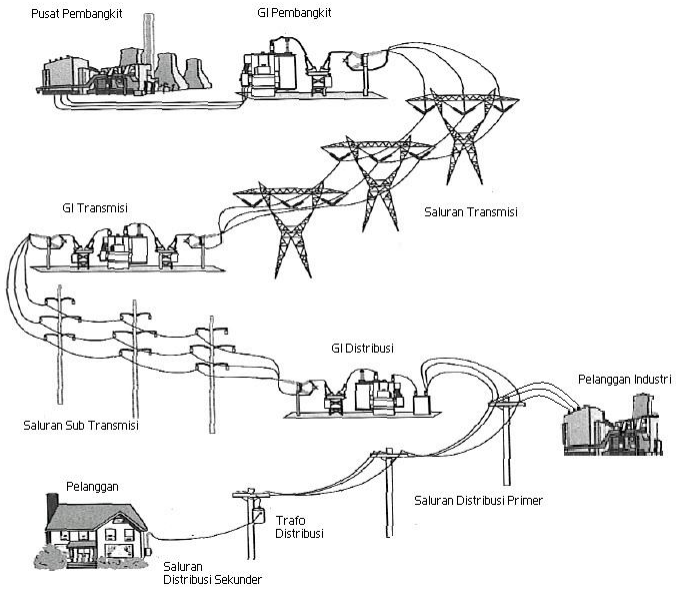
1.4 Manfaat

Manfaat yang diharapkan dari penulisan skripsi ini adalah dapat memberikan jadwal operasi sistem pembangkit listrik optimal dan dapat menjadi masukan bagi perusahaan listrik dalam manajemen operasionalnya sehingga diharapkan menghemat biaya operasional terutama biaya bahan bakar.

BAB 2. TINJAUAN PUSTAKA

2.1 Distribusi Tenaga Listrik ke Pelanggan

Tenaga listrik dibangkitkan dalam pusat-pusat listrik seperti PLTA, PLTGU, PLTG, PLTP, dan PLTD. Kemudian disalurkan ke Gardu Induk (GI) setelah dinaikkan terlebih dahulu tegangannya. Setelah tenaga listrik sampai di GI, tegangannya diturunkan menjadi tegangan menengah atau tegangan distribusi primer. Jaringan setelah keluar dari GI biasa disebut jaringan distribusi. Setelah tenaga listrik disalurkan melalui jaringan distribusi primer maka kemudian tenaga listrik diturunkan tegangannya dalam gardu-gardu distribusi menjadi tegangan rendah baru kemudian disalurkan melalui jaringan tegangan rendah untuk selanjutnya disalurkan ke rumah-rumah pelanggan (konsumen). Ilustrasi distribusi tenaga listrik tersebut disajikan pada Gambar 2.1 di bawah ini.



Gambar 2.1 Ilustrasi distribusi tenaga listrik ke pelanggan (Sumber : Imaduddin, 2008)

Pelanggan yang mempunyai daya tersambung besar langsung mendapatkan tenaga listrik dari jaringan tegangan menengah bahkan ada yang langsung tersambung dengan jaringan tegangan tinggi.

Berdasarkan uraian di atas dapat demengerti bahwa besar kecilnya konsumsi tenaga listrik ditentukan sepenuhnya oleh para pelanggan. Tergantung bagaimana pelanggan menggunakan alat-alat listriknya. Oleh karena itu perusahaan listrik harus menyesuaikan daya listrik yang dibangkitkan dari waktu ke waktu dengan kebutuhan pelanggan (Marsudi, 2006: 3).

2.2 *Unit Commitment (UC)*

Unit adalah satuan pembangkit tenaga listrik. *Commit* dari suatu unit pembangkit adalah mengoperasikan unit pembangkit tersebut sehingga berada dalam kondisi *on* (nyala). Pada kondisi tersebut, unit pembangkit akan dihubungkan dan disinkronkan dengan sistem, sehingga dapat mengirimkan tenaga listrik pada jaringan.

Masalah mengoperasikan satu atau beberapa unit pembangkit adalah salah satu masalah optimasi. Sangat tidak ekonomis bila terlalu banyak unit dioperasikan sementara unit pembangkit tersebut tidak dibutuhkan. Kombinasi yang tepat dalam mengoperasikan unit pembangkit akan menghemat biaya.

Tujuan dari *Unit Commitment (UC)* adalah penjadwalan unit pembangkit yang dioperasikan untuk melayani beban sistem. Dengan sejumlah keputusan untuk menghidupkan atau mematikan unit-unit pembangkit sesuai keperluan dengan memperhatikan beban sistem dan yang diperlukan tanpa melebihi kemampuan kapasitas pembangkit yang ada (Wood dan Wollenberg, 1996: 131).

2.3 *Economic Dispatch (ED)*

Bagaimana menentukan besarnya daya yang harus dibangkitkan oleh tiap unit pembangkit yang beroperasi agar dapat menyuplai kebutuhan tenaga listrik pelanggan disebut dengan *economic dispatch (ED)*. Dengan kata lain, jika *UC* menentukan unit-unit pembangkit yang harus beroperasi maka *ED* adalah masalah menentukan

besarnya daya yang harus dibangkitkan oleh tiap unit yang beroperasi tersebut (Wood dan Wollenberg, 1996: 57).

2.4 Kendala dalam *UC*

Kendala utama dalam *UC* adalah unit-unit pembangkit yang *on* harus dapat mencukupi kebutuhan beban sistem. Bentuk matematisnya sebagai berikut:

$$\sum_{i=1}^{N} P_i(t) = L(t), \tag{2.1}$$

dengan :

$P_i(t)$: daya keluaran unit ke-i pada waktu ke-t (MW) dan

$L(t)$: beban sistem pada waktu ke-t (MW)

(Wood dan Wollenberg, 1996: 32).

2.5 Biaya *Start-up*

Biaya *start-up* adalah biaya yang dibutuhkan untuk mengaktifkan suatu unit pembangkit. Rumusan matematis dari biaya *start-up* ini adalah :

$$StartCost(t) = \sum_{i=1}^{N} U_i^t (1 - U_i^{t-1}) S_i^t, \tag{2.2}$$

dengan :

$StartCost(t)$ = biaya *start-up* pada waktu *t*,

u_i^t = keadaan unit pembangkit ke-i untuk waktu sekarang (0 jika *off*, 1 jika *on*),

u_i^{t-1} = keadaan unit pembangkit ke-i untuk waktu sebelumnya (0 jika *off*, 1 jika *on*),

S_i^t = biaya *start-up* unit ke-I, dan

N = jumlah unit pembangkit yang *on*

(Dispankar dan Douglas, 1993:5).

2.6 Persamaan Biaya Konsumsi Bahan Bakar dan Pembangkitan

Persamaan konsumsi bahan bakar diperoleh dari persamaan panas rata-rata yang dijabarkan dalam bentuk polinomial kuadratik yaitu :

$$HR(P_i) = c + bP_i + aP_i^2, \quad (2.3)$$

dengan P_i adalah output dengan satuan MW (per unit) dengan batasan sebagai berikut:

$$(P_i)_{min} \leq P_i \leq (P_i)_{max}, \quad (2.4)$$

dengan $(P_i)_{min}$, $(P_i)_{max}$ adalah daya minimum dan daya maksimum dari unit ke-i (MW) dan *a, b, c* adalah koefisien konstanta persamaan panas rata-rata dari masing-masing unit pembangkit (Wood dan Wollenberg, 1996 : 10).

Sementara itu, untuk mengetahui besarnya biaya konsumsi bahan bakar diperoleh dengan mengalikan persamaan (2.3) dengan harga bahan bakar *(bb)*.

$$F(P_i) = HR(P_i).(bb) \quad (2.5)$$

Biaya pembangkitan (fungsi objektif yang akan diminimalisasi) diperoleh dengan menjumlahkan biaya konsumsi bahan bakar dan *startcost*. Yakni dengan menjumlahkan persamaan (2.2) dan persamaan (2.5) diperoleh:

$$FT = F(P_i) + Startcost(t) \quad (2.6)$$

(Dispankar dan Douglas, 1993: 5).

2.7 Algoritma Genetika

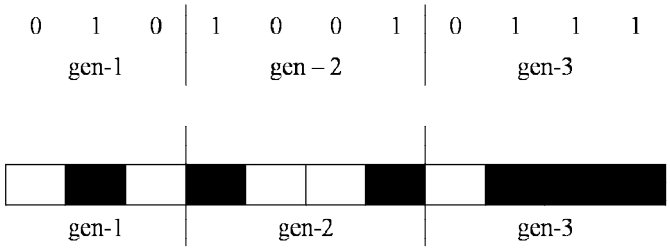
Algoritma genetika adalah algoritma pencarian yang didasarkan pada mekanisme seleksi alamiah dan genetika alamiah. Kemunculan algoritma genetika diinspirasikan dari teori-teori dalam ilmu biologi, sehingga banyak istilah dan konsep biologi yang digunakan (Suyanto, 2005: 1).

Pada algoritma genetika, teknik pencarian dilakukan sekaligus atas sejumlah solusi yang mungkin yang dikenal dengan istilah populasi. Individu yang terdapat dalam satu populasi disebut dengan istilah kromosom. Kromosom ini merupakan

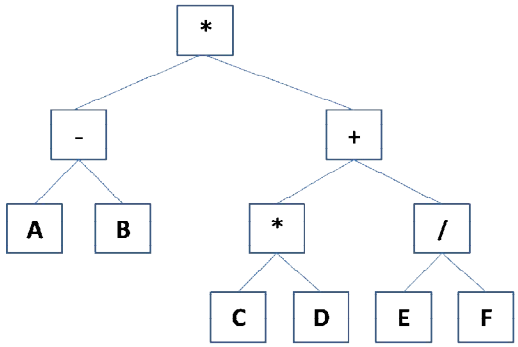
suatu solusi yang masih berbentuk simbol. Populasi awal dibangun secara acak, sedangkan populasi berikutnya merupakan hasil evolusi kromosom-kromosom melalui iterasi yang disebut generasi. Pada setiap generasi, kromosom akan melalui proses evaluasi dengan menggunakan alat ukur yang disebut dengan fungsi *fitness*. Nilai *fitness* dari suatu kromosom akan menunjukkan kualitas dari kromosom dalam populasi tersebut. Generasi berikutnya dikenal dengan istilah anak (*offspring*) terbentuk dari gabungan dua kromosom generasi sekarang yang bertindak sebagai induk (*parent*) dengan menggunakan operator *crossover* (penyilangan). Selain operator *crossover*, suatu kromosom dapat juga dimodifikasi dengan menggunakan operator mutasi. Populasi generasi yang baru dibentuk dengan cara menyeleksi nilai *fitness* dari kromosom induk dan nilai *fitness* dari kromosom anak, serta menolak kromosom-kromosom yang lainnya sehingga ukuran populasi konstan. Setelah melalui beberapa generasi, maka algoritma ini akan konvergen ke kromosom terbaik (Sanjoyo, 2006 : 5).

2.7.1 Skema Pengkodean

Pengkodean adalah cara untuk merepresentasikan masalah ke dalam bentuk kromosom. Kromosom tersebut harus membawa informasi dari solusi yang direpresentasikan. Ada beberapa macam pengkodean dalam algoritma genetika, diantaranya adalah pengkodean biner, pohon, array, bilangan real, daftar aturan, elemen permutasi dan elemen program (Kusumadewi, 2003:281). Contoh pengkodean biner dan pengkodean pohon disajikan pada Gambar 2.2 di bawah ini.



(a)



(b)

Gambar 2.2 Contoh pengkodean biner (a) dan pengkodean pohon (b)

2.7.2 Fungsi *Fitness*

Menurut Sanjoyo (2006:6) suatu individu dievaluasi berdasarkan suatu fungsi tertentu sebagai ukuran performansinya. Fungsi ini dinamakan fungsi *fitness*. Di dalam evolusi alam, individu yang bernilai *fitness* tinggi yang akan bertahan hidup. Sedangkan individu yang bernilai *fitness* rendah akan mati. Dalam algoritma genetika, fungsi *fitness* adalah fungsi objektif dari masalah yang akan dioptimasi. Fungsi ini sebagai ukuran keuntungan yang ingin dimaksimalkan atau sebagai ukuran biaya yang ingin diminimumkan.

2.7.3 Pemetaan Fungsi Objektif ke Fungsi *Fitness*

Dalam algoritma genetika, kromosom yang memiliki nilai *fitness* besar akan dipilih dalam proses seleksi. Sementara itu, dalam kasus minimasi, solusi yang diinginkan adalah kromosom yang memberikan biaya minimal. Oleh karena itu, perlu adanya pemetaan dari fungsi objektif (dalam kasus minimasi) menjadi fungsi *fitness*. Menurut Golberg (1989:76), pemetaan ini dirumuskan dengan:

$$f(x) = \begin{cases} C_{max} - g(x) \; jika \; g(x) \; < C_{max} \\ \\ 0 \; jika \; g(x) \; = C_{max} \end{cases} \tag{2.7}$$

dengan $f(x)$ adalah fungsi *fitness* dan $g(x)$ adalah fungsi objektif yang akan diminimasi. C_{max} adalah nilai terbesar $g(x)$ yang didapatkan pada populasi yang sedang berlangsung.

2.7.4 *Fitness Scaling*

Untuk meningkatkan performa algoritma genetika, dilakukan *fitness scaling*. Dalam algoritma genetika, sebuah populasi akan konvergen menuju solusi yang diinginkan, sehingga selisih dari nilai *fitness* antar kromosom dari generasi ke generasi akan menjadi sangat kecil. Selisih yang sangat kecil ini menghalangi kromosom terbaik untuk lebih unggul atau terpilih pada proses seleksi. *Fitness scaling* menyelesaikan masalah ini dengan mengatur nilai *fitness* agar menjadi paling superior (Ladd, 1996: 16).

Dalam skripsi ini akan digunakan *exponential scaling.* Dengan *exponential scaling,* kemampuan reproduksi dari kromosom dengan nilai *fitness* yang rendah akan meningkat dan superioritas dari kromosom terbaik akan bertambah. Rumusan *exponential scaling* sebagai berikut :

$$fitness(i) = (fitness(i) + 1)^2. \tag{2.8}$$

2.7.5 Operator Genetika

Proses reproduksi kromosom baru untuk menghasilkan kromosom terbaik dilakukan oleh beberapa operator genetika.

a. Seleksi

Operator seleksi menentukan kromosom mana yang dipilih untuk direproduksi. Probabilitas terpilihnya satu kromosom untuk direproduksi adalah sebesar nilai *fitness* tersebut dibagi dengan jumlah nilai *fitness* seluruh kromosom dalam populasi. Secara matematis, dinyatakan dengan persamaan:

$$p_i = \frac{f_i}{\sum_i f_i}, i = 1, \dots, n \tag{2.9}$$

dengan :

n : jumlah kromosom dalam suatu populasi,

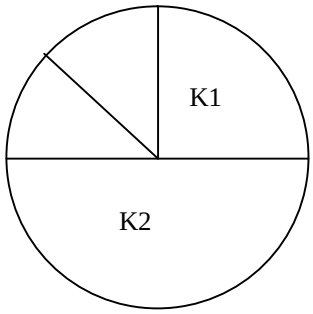
p_i : peluang kromosom i terpilih,

f_i : nilai *fitness* kromosom i , dan

$\sum_i f_i$: jumlah nilai *fitness* seluruh kromosom i.

Kromosom dengan *fitness* terbesar akan terpilih untuk direproduksi. Metode seleksi yang umum digunakan adalah *roulette-wheel* (roda *roulette*). Metode ini menirukan permainan *roulette-wheel* di mana masing-masing kromosom menempati potongan lingkaran pada roda *roulette* secara proporsional sesuai nilai *fitness*nya. Kromosom yang memiliki nilai *fitness* lebih besar, menempati potongan lingkaran yang lebih besar dibandingkan dengan kromosom bernilai *fitness* rendah (Suyanto, 2005: 13). Contoh seleksi dengan *roulette-wheel* disajikan pada Gambar 2.3 di bawah ini.

Kromosom	Nilai Fitness
K1	1
K2	2
K3	0.5
K4	0.5
Jumlah	4

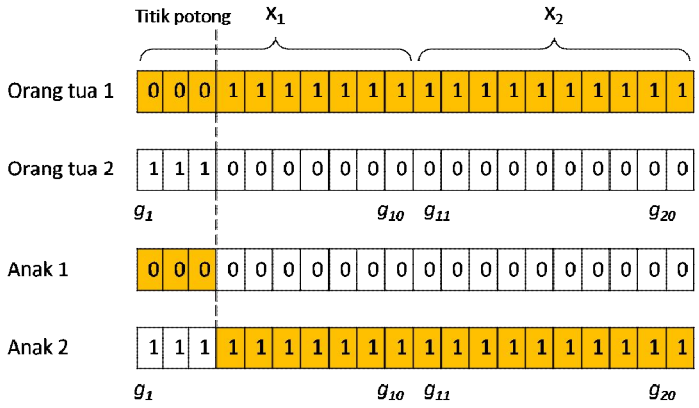


Gambar 2.3 Contoh seleksi dengan *roulette-wheel*

Metode *roulette-wheel selection* sangat mudah diimplementasikan dalam pemrograman. Pertama, dibuat interval kumulatif (dalam interval [0,1]) dari nilai *fitness* masing-masing kromosom dibagi total nilai *fitness* dari semua kromosom. Sebuah kromosom akan terpilih jika bilangan random yang dibangkitkan berada dalam interval akumulatifnya. Pada Gambar 2.3 di atas, K1 menempati interval nilai kumualtif [0;0,25], K2 berada dalam interval (0,25;0,75], K3 menempati interval (0,75;0,875] dan K4 dalam interval (0,875;1).

b. *Crossover*

Salah satu komponen penting dalam algoritma genetika adalah *crossover* atau pindah silang. Sebuah kromosom yang mengarah pada solusi yang bagus bisa diperoleh dari proses memindahsilangkan dua buah kromosom (Suyanto, 2005:13). Ilustrasi *crossover* terdapat pada Gambar 2.4 berikut.

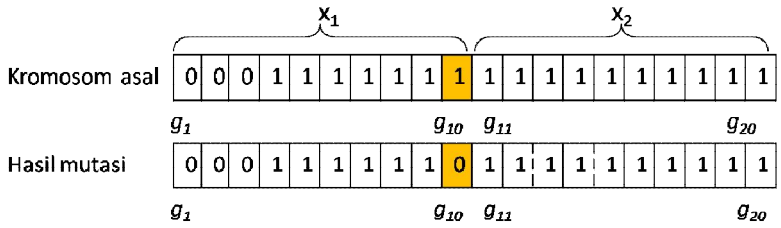


Gambar 2.4 Ilustrasi *one-point crossover*

c. Mutasi

Prosedur mutasi sangat sederhana. Untuk semua gen yang ada, jika bilangan random yang dibangkitkan kurang dari probabilitas mutasi p_m yang ditentukan maka ubah gen tersebut menjadi nilai kebalikannya (Suyanto, 2005:14). Dalam

pengkodean biner, 0 diubah 1 dan 1 diubah 0. Gambar 2.5 mengilustrasikan proses mutasi yang terjadi pada gen g_{10}.



Gambar 2.5 Contoh mutasi

d. Penentuan Parameter Genetika

Pada algoritma genetika, parameter digunakan untuk memberikan suatu probabilitas (kemungkinan atau peluang) yang berupa persentase. Probabilitas yang ada dalam algoritma ini adalah probabilitas *crossover* dan probabilitas mutasi.

Probabilitas *crossover* (p_c), menunjukkan jumlah kromosom yang mengalami proses tukar silang dalam satu populasi. Jika *crossover* tidak ada, maka anak akan meniru induknya. Tetapi jika *crossover* ada, anak dibuat dari bagian kromosom induk (jika kemungkinan *crossover* adalah 100%, maka semua anak dibuat oleh *crossover*, tetapi jika 0% maka anak dibuat dengan meniru kromosom dari populasi lama secara keseluruhan).

Probabilitas mutasi (p_m), menunjukkan jumlah kromosom yang mengalami proses mutasi dalam satu populasi. Jika mutasi ada, bagian kromosom diubah (jika kemungkinannya 100%, seluruh kromosom akan diubah dan jika 0% tidak ada yang diubah), tetapi jika tidak ada mutasi, anak langsung mengambil dari *crossover* dengan tidak mengubah lagi.

Menurut Suyanto (2005:43) ada beberapa ketentuan parameter algoritma genetika yang optimal. Untuk ukuran populasi, berkisar antara 30 sampai 1000. Ukuran populasi yang terlalu kecil akan menyebabkan algoritma genetika cepat

konvergen karena rendahnya variasi pada kromosom-kromosom dalam populasi. Tetapi jika ukuran populasi terlalu besar maka proses algoritma genetika akan menjadi lambat. Probabilitas crossover ditentukan antara 0,6 sampai 0,9 dan berkebalikan dengan probabilitas crossover, probabilitas mutasi biasanya sangat kecil, nilainya sekitar 1 dibagi dengan jumlah gen dalam satu populasi. Sehingga peluang mutasi hanya terjadi pada sekitar satu gen saja.

BAB 3. METODE PENELITIAN

3.1 Data Penelitian

Data yang digunakan dalam penelitian ini adalah karakteristik pembangkit meliputi daya maksimum dan mimimum, *start cost*, karakteristik *heatrate* (konstanta *a*, *b*, *c*), jenis dan harga bahan bakar dan beban harian. Pembangkit yang akan diteliti adalah pembangkit pada PT. PLN PJB II yang beroperasi di Jawa Timur dan Bali. Data selengkapnya dapat dilihat lampiran A.

Data yang digunakan merupakan data sekunder yang diperoleh dari penelitian sebelumnya yakni tugas akhir Optimasi Biaya pada Penjadwalan Pembangkit Menggunakan Metode Pemrograman Dinamis *Fuzzy* (Wibowo : 2003).

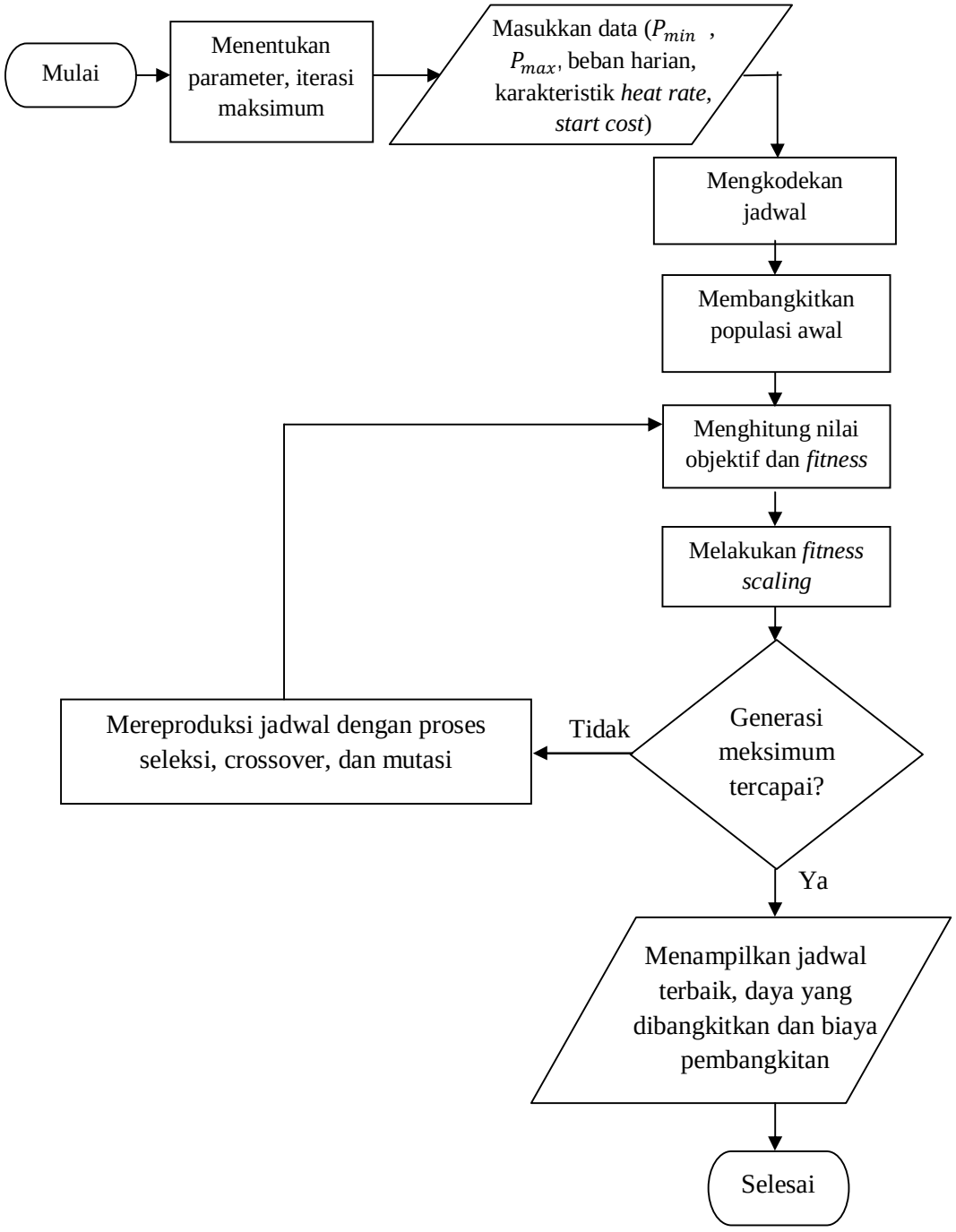
3.2 Langkah-langkah Penyelesaian

Untuk dapat memahami penyelesaian permasalahan dengan algoritma genetika, diperlukan adanya transfomasi istilah dari masalah penjadwalan operasi sistem pembangkit tenaga listrik *(unit commitment)* ke dalam algoritma genetika yang dapat dilihat pada Tabel 3.1 berikut ini.

Tabel 3.1 Istilah *unit commitment* dalam algoritma genetika

Unit commitment	Algoritma genetika
jadwal pembangkitan	kromosom
unit pembangkit	gen
kondisi unit pembangkit	allele
sekumpulan jadwal dalam satu iterasi	populasi

Penelitian ini akan menggunakan dua tahapan penyelesaian yakni penyelesian manual dan penyelesaian dengan program mengikuti langkah-langkah yang disajikan dalam Gambar 3.1 di bawah ini.



Gambar 3.1 Diagram alir penyelesaian optimasi penjadwalan sistem pembangkit tenaga listrik

BAB 4. HASIL DAN PEMBAHASAN

Pada bab ini akan disajikan hasil dan pembahasan penyelesaian masalah penjadwalan sistem pembangkit tenaga listrik menggunakan algoritma genetika dengan penambahan proses *fitness scaling.*

4.1 Hasil

4.1.1. Penyelesaian Manual dengan Input 3 (Tiga) Unit Pembangkit

Penyelesaian manual dengan 3 unit pembangkit dimaksudkan untuk menjelaskan cara kerja algoritma genetika dalam menyelesaikan masalah penjadwalan operasi sistem pembangkit tenaga listrik. Berikut langkah-langkahnya:

a. Penentuan Parameter dan Iterasi maksimum

Beberapa parameter genetika yang harus ditentukan adalah *popsize*, peluang *crossover* (p_c), peluang mutasi (p_m) dan kriteria berhenti (iterasi maksimum). *Popsize* dalam penyelesaian manual ini ditentukan sebanyak 4 (empat) artinya ada empat jadwal dalam satu populasi. Menurut Suyanto (2005:43) probabilitas *crossover* (p_c) ditentukan antara 0,6 sampai 0,9. Dalam penyelesaian manual ini p_c yang digunakan adalah 0,6. Sementara nilai probabilitas mutasi sekitar 1 dibagi jumlah gen dalam satu populasi sehingga $p_m = 0{,}083$.

b. Data Masukan

Data masukan berupa nama unit pembangkit, daya minimum (P_{min}) dan maksimum (P_{mak}) yang bisa dibangkitkan oleh tiap unit pembangkit, karakteristik *heatrate (H)*, harga bahan bakar dan biaya *start-up* tiap unit. Data-data tersebut disajikan dalam Tabel 4.1.

Tabel 4.1 Karakteristik pembangkit

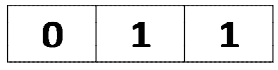
No.	Nama Pembangkit	P_{mak} (MW)	P_{min} (MW)	Startcost (juta Rupiah)	Bahan bakar	Harga bahan bakar (Rp/Mkal)	H
1.	PLTU Paiton 1	400	225	344,94	Batu bara	42,589	$0{,}2P^2 + 2467{,}4P + 6000$
2.	PLTU Gresik 1	100	43	72,6	Gas	96,1636	$1{,}0P^2 + 2781{,}4P + 11250$
3.	PLTU Gresik 3	200	66	115,91	Gas	96,1636	$0{,}01P^2 + 2496{,}1P + 125250$

Mkal (mega kalori) adalah satuan untuk energi dan kalor.

Beban yang harus disuplai adalah 350 MW.

c. Pengkodean dalam Bentuk Kromosom

Dalam menyelesaikan masalah optimasi penjadwalan sistem pembangkit tenaga listrik, teknik pengkodean yang digunakan adalah pengkodean biner yakni menggunakan angka nol (0) dan satu (1). Gen merepresentasikan unit i ($i = 1{,}2{,}3$) dan kromosom merepresentasikan jadwal pembangkitan. Allele berupa nilai 0 atau 1. Kode 0 artinya unit tidak beroperasi, kode 1 artinya unit beroperasi. Pengkodean biner untuk masalah penjadwalan operasi sistem pembangkit listrik dengan tiga unit dimana unit 1 tidak beroperasi sementara unit 2 dan 3 beroperasi dapat dilihat pada Gambar 4.1 di bawah ini.



Gambar 4.1 Kromosom dengan pengkodean biner

Pada Gambar 4.1 di atas allele gen pertama bernilai 0 sementara allele gen kedua dan ketiga bernilai 1 artinya unit 1 tidak beroperasi, sementara unit 2 dan 3 beroperasi.

d. Pembangkitan Populasi Awal

Pada tahapan ini dibangkitkan empat jadwal pembangkitan sebanyak ukuran populasi. Empat jadwal untuk populasi awal disajikan pada Tabel 4.2 di bawah ini.

Tabel 4.2 Empat jadwal untuk populasi awal

Jadwal ke-	Pengkodean		
	Unit 1	Unit 2	Unit 3
1	1	0	1
2	1	1	1
3	1	1	0
4	1	0	0

Sementara itu, daya yang dibangkitkan oleh masing-masing unit disajikan pada Tabel 4.3 berikut.

Tabel 4.3 Daya yang dibangkitkan oleh masing-masing unit

Jadwal Ke-	P_1	P_2	P_3
1	225	0	125
2	241	43	66
3	250	100	0
4	350	0	0

e. Perhitungan Nilai Objektif & *Fitness*

Langkah selanjutnya adalah menghitung nilai objektif masing-masing jadwal dan nilai *fitness*nya. Nilai objektif diperoleh dengan memasukkan data P_i atau daya yang dibangkitkan oleh unit ke-i yang tercantum pada Tabel 4.3, harga bahan bakar dan *startcost* yang tercantum pada Tabel 4.1 ke persamaan (2.6) pada Bab 2.

Untuk jadwal ke-1,

$$F_1 = c + bP_1 + aP_1^2(bb) + U_1^t\left(1 - U_1^{t-1}\right) S_1^t$$

$$= (6000 + 2467{,}4P_1 + 0{,}2P_1^2)(42{,}589) + 1(1 - 0)344.940.000$$

$$= (6000 + 2467{,}4(225) + 0{,}2(225)^2)(42{,}589) + 344.940.000$$

$$= 24.330.669{,}81 + 344.940.000$$

$$F_1 = 369.270.669{,}8$$

$F_2 = 0$

$F_3 = c + bP_3 + aP_3^2(bb) + U_3^t\left(1 - U_3^{t-1}\right)S_3^t$

$= (125250 + 2496{,}1P_3 + 0{,}01P_3^2)96{,}1636 + 1(1 - 0)115.910.000$

$= (125250 + 2496{,}1(125) + 0{,}01(125)^2)96{,}1636 + 115.910.000$

$= 42.063.761{,}71 + 115.910.000$

$F_3 = 157.973.761{,}7$

Total biaya pembangkitan diperoleh dengan menjumlahkan F_1, F_2 dan F_3.

$F_1 + F_2 + F_3 = 527.244.431{,}5$

Biaya pembangkitan atau nilai objektif dari jadwal ke-1 telah diperoleh yakni sebesar Rp 527.244.431,5 –. Dengan cara yang sama, akan diperoleh nilai objektif untuk jadwal ke-2 dan seterusnya. Hasilnya dapat dilihat pada Tabel 4.4 berikut.

Tabel 4.4 Nilai objektif masing-masing jadwal pembangkitan

Jadwal Ke-	F_1	F_2	F_3	$F_1 + F_2 + F_3$
1	369.270.669,8	0	157.973.761,7	527.244.431,52
2	371.015.524,10	85.360.832,79	143.800.921,28	600.177.278,17
3	371.998.921,2	101.390.420,2	0	473.389.341,35
4	383.018.399,01	0	0	383.018.399,01

Dari Tabel 4.4 diketahui nilai objektif terbesar adalah Rp 600.177.278,17 yang diperoleh pada jadwal ke-2. Setelah itu, nilai objektif dipetakan menjadi nilai *fitness* sesuai dengan persamaan (2.7). Perhitungan dan hasil pemetaan disajikan dalam Tabel 4.5 berikut.

Tabel 4.5 Pemetaan nilai objektif menjadi nilai *fitness*

Jadwal ke-	$F_1 + F_2 + F_3$	*Fitness*
1	527.244.431,52	600.177.278,17 - 527.244.431,52 = 72.932.846,65
2	600.177.278,17	600.177.278,17 - 600.177.278,17 = 0
3	473.389.341,35	600.177.278,17 – 473.389.341,35 = 126.787.936,82
4	383.018.399,01	600.177.278,17 – 383.018.399,01 = 217.158.879,16

f. Perhitungan *Fitness* Scaling

Selanjutnya dihitung hasil *fitness scaling* dari nilai *fitness* masing-masing jadwal sesuai dengan persamaan (2.8). Hasil dan perhitungan dapat dilihat pada Tabel 4.6 berikut.

Tabel 4.6 Nilai *fitness scaling* masing-masing jadwal pembangkitan

Jadwal ke-	Nilai *fitness*	*Fitness scaling* $(fitness(i) + 1)^2$
1	72.932.846,65	5.319.200.266.600.080,00
2	0	1,00
3	126.787.936,82	16.075.181.175.455.400,00
4	217.158.879,16	47.157.979.232.039.500,00

Nilai *fitness scaling* inilah yang akan digunakan pada proses selanjutnya.

g. Reproduksi jadwal pembangkitan

1) Seleksi

Metode seleksi yang digunakan adalah *roulette wheel* dengan langkah-langkah:

a) Menghitung total nilai *fitness* dengan rumus $\sum_{i=1}^{4} f_i = f_1 + f_2 + f_3 + f_4$

b) Menghitung probabilitas & nilai kumulatif tiap jadwal pembangkitan.

Probabilitas masing-masing jadwal pembangkitan diperoleh dengan persamaan (2.9).

$$p_i = \frac{f_i}{\sum_i f_i}, i = 1, \dots, n$$

Nilai probabilitas masing-masing jadwal dapat dilihat pada tabel 4.7 berikut.

Tabel 4.7 Nilai probabilitas & kumulatif tiap kromosom

Jadwal ke-	f_i	p_i	Kumulatif
1	5.319.200.266.600.080,00	0.077593247	0.077593247
2	1,00	0.000000000	0.077593247
3	16.075.181.175.455.400,00	0.234494932	0.312088179
4	47.157.979.232.039.500,00	0.687911821	1.000000000
Jumlah	68.552.360.674.095.000,00	1	

c) Membangkitkan bilangan acak untuk seleksi (r_s) antara 0 – 1 sebanyak ukuran populasi. Bilangan acak untuk seleksi yang digunakan dalam skripsi ini disajikan dalam Tabel 4.8 di bawah ini.

Tabel 4.8 Bilangan acak untuk seleksi (r_s)

No.	r_s
1	0,314
2	0,461
3	0,342
4	0,742

d) Memilih jadwal ke – π sebagai induk dengan syarat $q_{\pi-1} < r_s < q_\pi$ dengan $\pi = 1,2,\dots,5$.

Untuk $r_1 = 0{,}314$ berada pada rentang $q_{3-2} < 0{,}314 < q_3$ sehingga jadwal 1 digantikan oleh jadwal 3.

Untuk $r_2 = 0{,}461$ berada pada rentang $q_{4-3} < 0{,}461 < q_4$ sehingga jadwal 2 digantikan oleh jadwal 4.

Untuk $r_3 = 0{,}342$ berada pada rentang $q_{4-3} < 0{,}342 < q_4$ sehingga jadwal 3 digantikan oleh jadwal 4.

Untuk $r_4 = 0{,}742$ berada pada rentang $q_{4-3} < 0{,}742 < q_4$ sehingga jadwal 4 tetap.

Populasi setelah seleksi dapat dilihat pada Tabel 4.9 berikut.

Tabel 4.9 Populasi setelah seleksi

Jadwal ke-	Populasi Sebelum Seleksi			Populasi setelah seleksi		
	Unit 1	Unit 2	Unit 3	Unit 1	Unit 2	Unit 3
1	1	0	1	1	1	0
2	1	1	1	1	0	0
3	1	1	0	1	0	0
4	1	0	0	1	0	0

2) *Crossover*

Teknik *Crossover* yang akan dipilih adalah *one point crossover* dengan langkah-langkah sebagai berikut:

a) Menentukan jumlah jadwal yang akan di*crossover*

Peluang *crossover* (p_c) sebesar $0{,}6$. Berarti 60% atau dua dari empat jadwal yang dihasilkan dari proses seleksi akan mengalami *crossover*.

b) Membangkitkan bilangan acak (r_c)

Bilangan acak (r_c) dibangkitkan untuk menentukan jadwal yang akan mengalami *crossover* dengan ketentuan jika $r_c < p_c$ maka jadwal ke-π terpilih sebagai induk. Bilangan acak untuk *crossover* disajikan dalam Tabel 4.10 berikut.

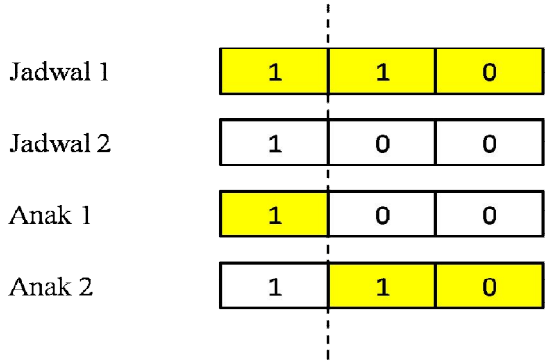
Tabel 4.10 Bilangan acak untuk *crossover*

Jadwal ke-	r_c
1	0,452
2	0,211
3	0,698
4	0,877

Berdasarkan Tabel 4.10, jadwal yang terpilih sebagai induk adalah jadwal ke-1 dan 2.

c) Melakukan *Crossover*

Crossover dilakukan dengan menentukan satu titik potong secara acak. Misalkan titik potong antara gen pertama dan kedua. Gen pertama pada jadwal 1 dipindahkan ke jadwal anak pertama dengan poisisi yang sama. Gen kedua dan ketiga dari jadwal 2 dipindahkan ke jadwal anak dengan posisi yang sama. Selanjutnya, gen pertama dari jadwal 2 dipindahkan ke jadwal anak kedua dengan posisi sama. Gen kedua dan ketiga dari jadwal 1 dipindahkan ke jadwal anak kedua dengan posisi sama. Proses *crossover* dan hasilnya dapat dilihat pada Gambar 4.2 di bawah ini.



Gambar 4.2 *Crossover* jadwal 1 dan 2

Sehingga populasi setelah *Crossover* adalah seperti yang disajikan pada Tabel 4.11 berikut.

Tabel 4.11 Populasi setelah *crossover*

Jadwal ke-	Sebelum *Crossover*			Setelah *Crossover*		
	Unit 1	Unit 2	Unit 3	Unit 1	Unit 2	Unit 3
1	1	1	0	1	0	0
2	1	0	0	1	1	0
3	1	0	0	1	0	0
4	1	0	0	1	0	0

3) Mutasi

Mutasi dilakukan dengan menentukan probabilitas mutasi (p_m) terlebih dahulu. Dalam penelitian ini $p_m = 0.083$ dengan jumlah gen = 12. Artinya 0.996 gen akan mengalami mutasi. Penentuan gen yang mengalami mutasi dilakukan dengan membangkitkan bilangan acak untuk mutasi (r_m) sebanyak gen. Apabila $r_m < p_m$ maka gen tersebut akan dimutasi.

Bilangan acak untuk mutasi (r_m) yang digunakan dalam skripsi ini disajikan pada Tabel 4.12 di bawah ini.

Tabel 4.12 Bilangan acak untuk mutasi

Jadwal ke-	Gen ke-	r_m
1	1	0,9450
	2	0,8891
	3	0,0862
2	4	0,4523
	5	0,9734
	6	0,9605
3	7	0,8146
	8	0,0637
	9	0,6328
4	10	0,0869
	11	0,1960
	12	0,7480

Berdasarkan Tabel 4.12 di atas dapat dilihat bahwa gen ke-8 pada jadwal ketiga, $r_m < p_m$ sehingga gen tersebut mengalami mutasi. Hal ini sesuai dengan banyaknya gen yang akan dimutasi yakni 0.996 gen atau satu gen. Dengan demikian, terjadi perubahan jadwal pembangkitan setelah mutasi. Populasi setelah mutasi dapat dilihat pada Tabel 4.13 dibawah ini.

Tabel 4.13 Populasi setelah mutasi

Jadwal ke-	Sebelum Mutasi			Setelah Mutasi		
	Unit 1	Unit 2	Unit 3	Unit 1	Unit 2	Unit 3
1	1	0	0	1	0	0
2	1	1	0	1	1	0
3	1	0	0	1	1	0
4	1	0	0	1	0	0

Proses algoritma genetika pada satu iterasi telah selesai. Jadwal dan nilai objektif (biaya pembangkitan) dalam satu iterasi disajikan pada Tabel 4.14 di bawah ini.

Tabel 4.14 Jadwal dalam satu iterasi

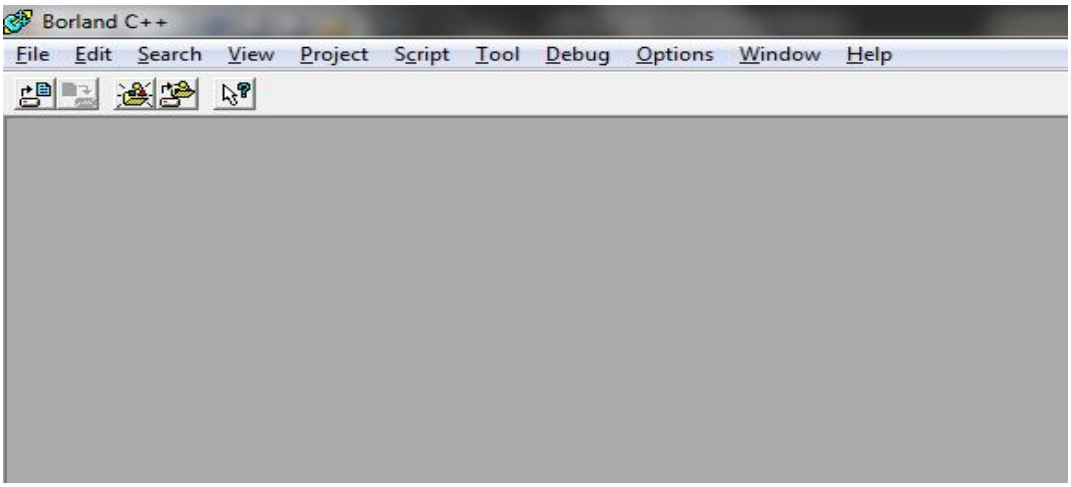
Jadwal ke-	Unit 1	Unit 2	Unit 3	Biaya Pembangkitan (Rp)
1	1	0	0	383.018.399,01
2	1	1	0	473.389.341,35
3	1	1	0	473.389.341,35
4	1	0	0	383.018.399,01

Proses ini dilakukan hingga iterasi mencapai maksimum sesuai dengan kriteria berhenti yakni apabila telah mencapai 1000 iterasi. Apabila kriteria berhenti tersebut belum terpenuhi maka dilanjutkan ke proses perhitungan nilai fitness, fitness scaling, reproduksi dengan seleksi, crossover dan mutasi sehingga diperoleh iterasi ke-2 hingga ke-1000.

4.1.2. Penyelesaian dengan program untuk input 15 unit pembangkit

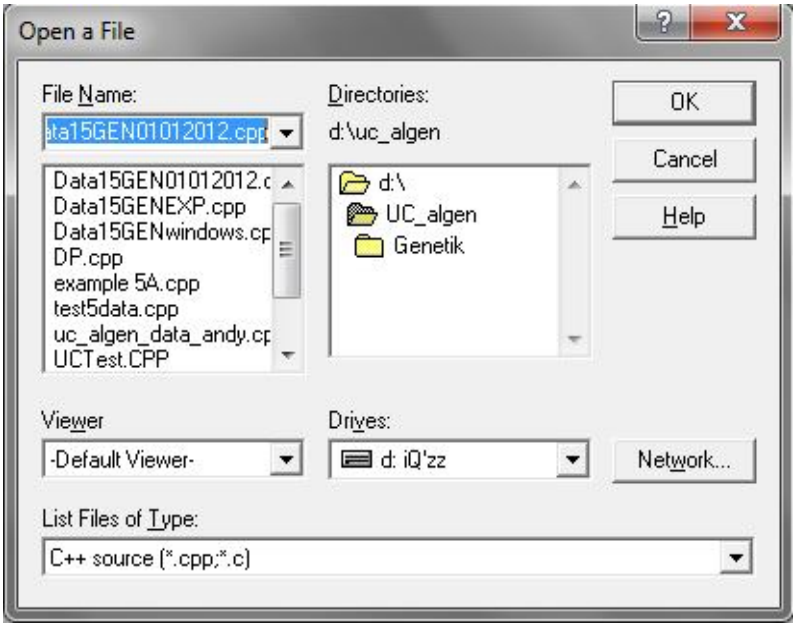
Penerapan algoritma genetika pada masalah penjadwalan operasi sistem pembangkit tenaga listrik dengan bantuan program dijelaskan melalui tahapan-tahapan berikut ini:

a. Buka jendela program C++ seperti Gambar 4.3 di bawah ini.



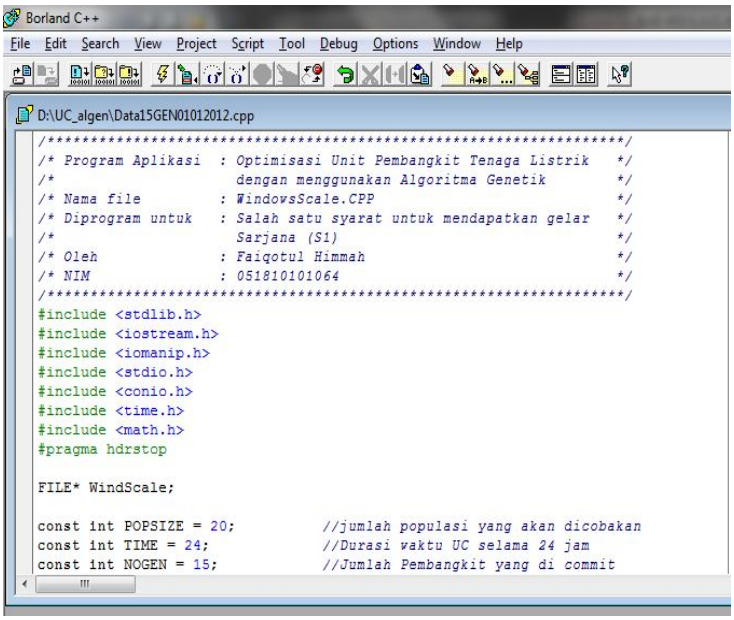
Gambar 4.3 Tampilan awal program borland C++

b. Melalui menu bar *File,* pilih *Open* lalu akan muncul tampilan seperti pada Gambar 4.4 di bawah ini.



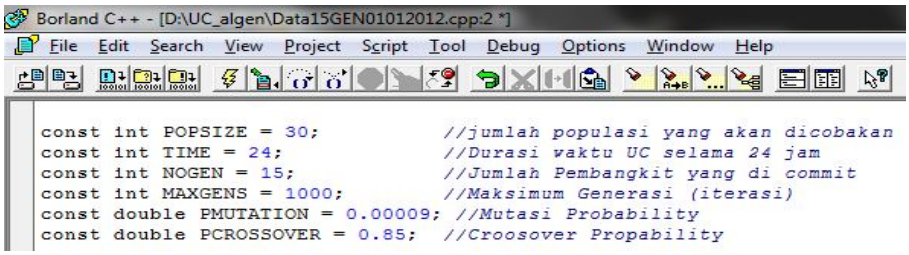
Gambar 4.4 Jendela *open a file*

c. Klik pada nama file data15GEN01012012.cpp lalu klik OK. Akan muncul tampilan program seperti pada Gambar 4.5 di bawah ini.



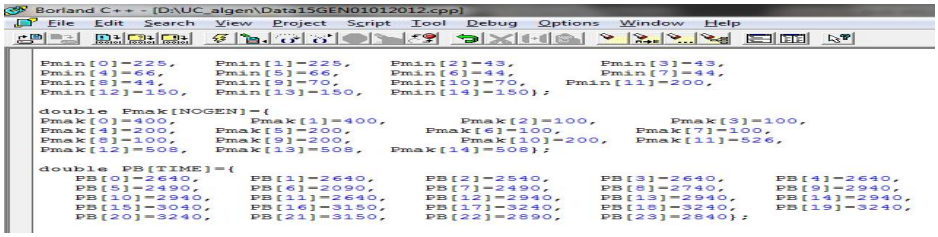
Gambar 4.5 Tampilan file program

d. Masukkan parameter genetika. Menurut Suyanto (2005:43) ukuran populasi *(popsize)* teroptimal adalah antara 30 hingga 1000. Dalam skripsi ini ditentukan ukuran populasi sebesar 30. Probaibiltas crossover (p_c) antara 0,6 hingga 0,9 dan probabilitas mutasi nilainya berkisar antara satu dibagi jumlah gen. Dalam skripsi ini digunakan p_m = 0,00009, p_c = 0,85, waktu penelitan = 24, banyaknya unit = 15 dan kriteria berhenti adalah setelah mencapai 1000 iterasi agar hasil yang diperoleh konvergen ke nilai terbaik. Apabila jumlah iterasi maksimum sangat sedikit, sangat memungkinkan masih ada nilai yang lebih baik dari yang diperoleh. Tampilan file program untuk input parameter genetika seperti Gambar 4.6 di bawah ini.



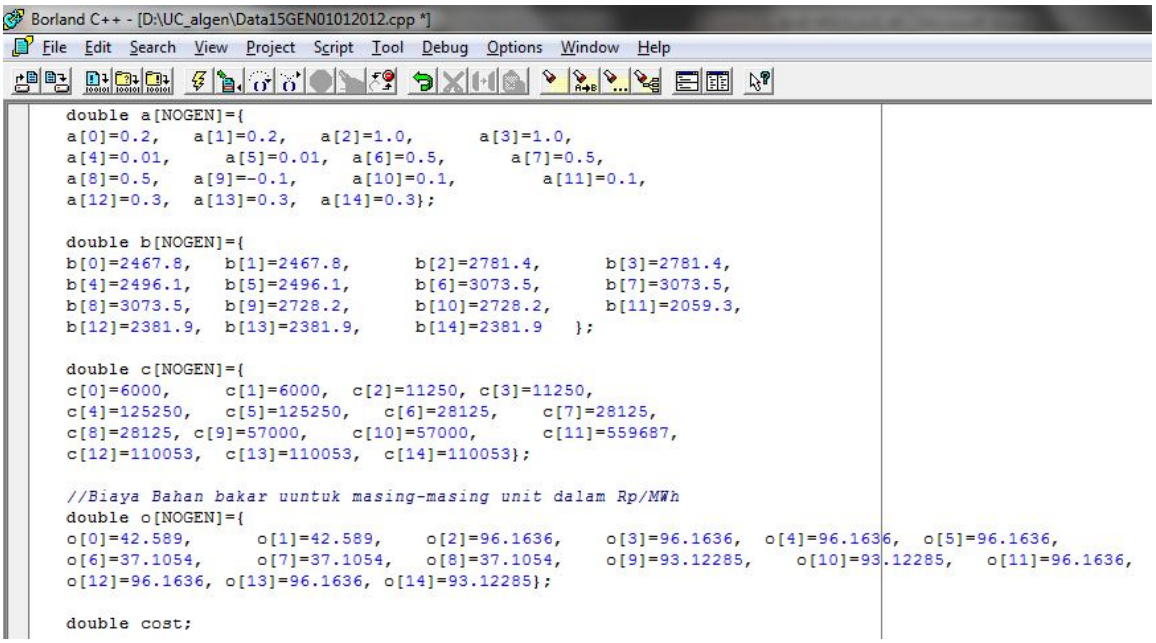
Gambar 4.6 Tampilan file program aplikasi untuk input parameter genetika

e. Masukkan data $P_{min,}P_{mak}$ masing-masing unit pembangkit dan data beban harian setiap jam selama 24 jam penelitian seperti pada Gambar 4.7 di bawah ini. Data yang digunakan terdapat pada lampiran A & lampiran B.



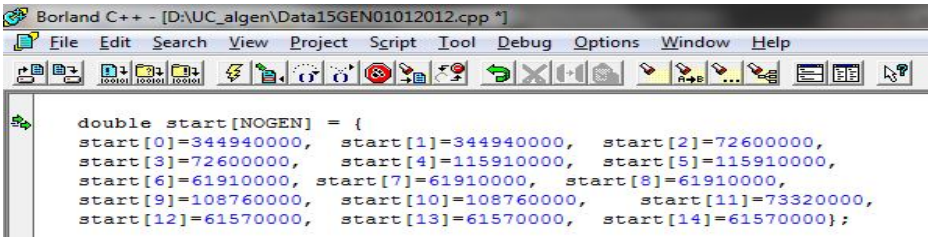
Gambar 4.7 Tampilan file program aplikasi untuk input data $P_{min,}P_{mak}$ & beban harian dalam 24 jam

f. Masukkan data karakteristik *heatrate* & harga bahan bakar masing-masing unit pembangkit seperti Gambar 4.8 berikut.



Gambar 4.8 Tampilan file program aplikasi untuk input data karakteristik *heatrate* dan harga bahan bakar

g. Masukkan data *startcost* masing-masing unit pembangkit seperti Gambar 4.9 di bawah ini.

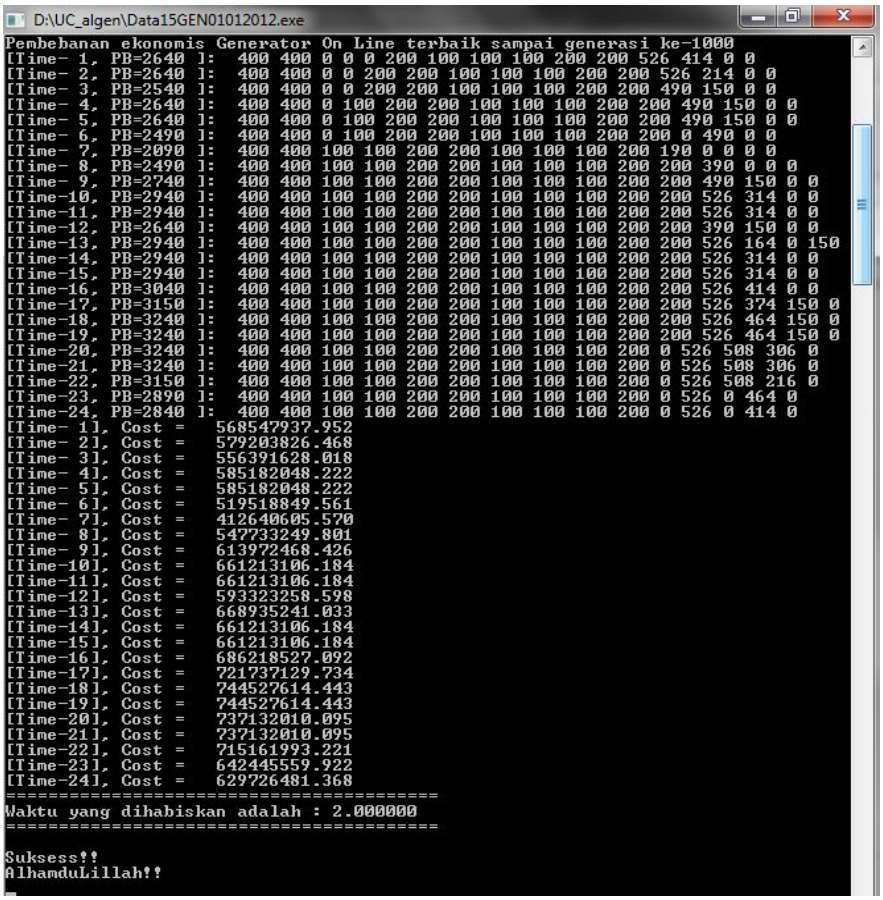


Gambar 4. 9 Tampilan file program aplikasi untuk input data *startcost* masing-masing unit pembangkit

h. Jalankan program, dengan langkah klik button run  seperti Gambar 4.10 di bawah ini.

Gambar 4.10 Klik button *run*

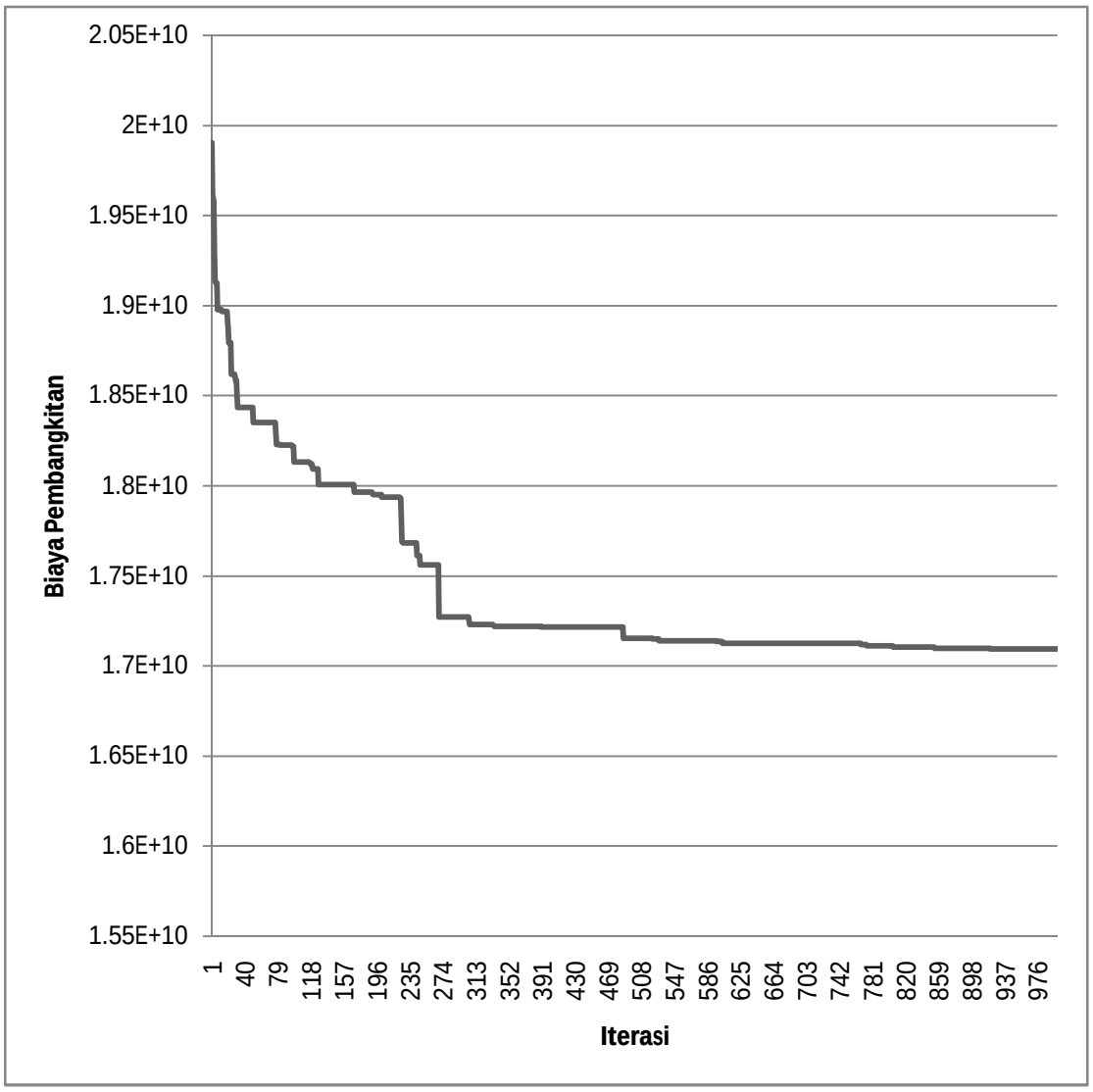
i. Hasilnya adalah seperti pada Gambar 4.11 berikut ini.



Gambar 4.11 Tampilan hasil program file aplikasi

Output program di atas adalah biaya pembangkitan terbaik dalam setiap iterasi, jadwal terbaik, daya yang dibangkitkan (*economic dispatch*) untuk jadwal terbaik, biaya pembangkitan tiap jam penelitian dan waktu yang dibutuhkan untuk *running* program.

Grafik biaya pembangkitan dari iterasi ke-1 hingga iterasi ke-1000 disajikan pada Gambar 4.13 di bawah ini.



Gambar 4.13 Grafik biaya pembangkitan tiap iterasi

Jadwal pembangkitan terbaik diperoleh pada iterasi ke-920 dengan biaya Rp 17.093.816.033,7454 dibulatkan menjadi Rp 17.093.816.034 (tujuh belas milyar sembilan puluh tiga juta delapa ratus enam belas ribu tiga puluh empat rupiah). Jadwal terbaik disajikan dalam Tabel 4.15 berikut ini.

Tabel 4.15 Jadwal terbaik

Jam	Jadwal Unit ke-														
ke-	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	*on*	*on*	*off*	*off*	*off*	*on*	*on*	*on*	*on*	*on*	*on*	*on*	*on*	*off*	*off*
2	*on*	*on*	*off*	*off*	*on*	*on*	*on*	*on*	*on*	*on*	*on*	*on*	*on*	*off*	*off*
3	*on*	*on*	*off*	*off*	*on*	*on*	*on*	*on*	*on*	*on*	*on*	*on*	*on*	*off*	*off*
4	*on*	*on*	*off*	*on*	*on*	*on*	*on*	*on*	*on*	*on*	*on*	*on*	*on*	*off*	*off*
5	*on*	*on*	*off*	*on*	*on*	*on*	*on*	*on*	*on*	*on*	*on*	*on*	*on*	*off*	*off*
6	*on*	*on*	*off*	*on*	*on*	*on*	*on*	*on*	*on*	*on*	*on*	*off*	*on*	*off*	*off*
7	*on*	*on*	*on*	*on*	*on*	*on*	*on*	*on*	*on*	*on*	*on*	*off*	*off*	*off*	*off*
8	*on*	*on*	*on*	*on*	*on*	*on*	*on*	*on*	*on*	*on*	*on*	*on*	*off*	*off*	*off*
9	*on*	*on*	*on*	*on*	*on*	*on*	*on*	*on*	*on*	*on*	*on*	*on*	*on*	*off*	*off*
10	*on*	*on*	*on*	*on*	*on*	*on*	*on*	*on*	*on*	*on*	*on*	*on*	*on*	*off*	*off*
11	*on*	*on*	*on*	*on*	*on*	*on*	*on*	*on*	*on*	*on*	*on*	*on*	*on*	*off*	*off*
12	*on*	*on*	*on*	*on*	*on*	*on*	*on*	*on*	*on*	*on*	*on*	*on*	*on*	*off*	*off*
13	*on*	*on*	*on*	*on*	*on*	*on*	*on*	*on*	*on*	*on*	*on*	*on*	*on*	*off*	*on*
14	*on*	*on*	*on*	*on*	*on*	*on*	*on*	*on*	*on*	*on*	*on*	*on*	*on*	*off*	*off*
15	*on*	*on*	*on*	*on*	*on*	*on*	*on*	*on*	*on*	*on*	*on*	*on*	*on*	*off*	*off*
16	*on*	*on*	*on*	*on*	*on*	*on*	*on*	*on*	*on*	*on*	*on*	*on*	*on*	*off*	*off*
17	*on*	*on*	*on*	*on*	*on*	*on*	*on*	*on*	*on*	*on*	*on*	*on*	*on*	*on*	*off*
18	*on*	*on*	*on*	*on*	*on*	*on*	*on*	*on*	*on*	*on*	*on*	*on*	*on*	*on*	*off*
19	*on*	*on*	*on*	*on*	*on*	*on*	*on*	*on*	*on*	*on*	*on*	*on*	*on*	*on*	*off*
20	*on*	*on*	*on*	*on*	*on*	*on*	*on*	*on*	*on*	*on*	*off*	*on*	*on*	*on*	*off*
21	*on*	*on*	*on*	*on*	*on*	*on*	*on*	*on*	*on*	*on*	*off*	*on*	*on*	*on*	*off*
22	*on*	*on*	*on*	*on*	*on*	*on*	*on*	*on*	*on*	*on*	*off*	*on*	*on*	*on*	*off*
23	*on*	*on*	*on*	*on*	*on*	*on*	*on*	*on*	*on*	*on*	*off*	*on*	*off*	*on*	*off*
24	*on*	*on*	*on*	*on*	*on*	*on*	*on*	*on*	*on*	*on*	*off*	*on*	*off*	*on*	*off*

Daya yang dibangkitkan *(economic dispatch)* oleh tiap unit yang beroperasi disajikan pada Tabel 4.16 berikut ini.

Tabel 4.16 *Economic dispatch* untuk kormosom terbaik

Jadwal ke-	*Economic Dispatch* (MW) Unit ke-														
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	400	400	0	0	0	200	100	100	100	200	200	526	414	0	0
2	400	400	0	0	200	200	100	100	100	200	200	526	214	0	0
3	400	400	0	0	200	200	100	100	100	200	200	490	150	0	0
4	400	400	0	100	200	200	100	100	100	200	200	490	150	0	0
5	400	400	0	100	200	200	100	100	100	200	200	490	150	0	0
6	400	400	0	100	200	200	100	100	100	200	200	0	490	0	0
7	400	400	100	100	200	200	100	100	100	200	190	0	0	0	0
8	400	400	100	100	200	200	100	100	100	200	200	390	0	0	0
9	400	400	100	100	200	200	100	100	100	200	200	490	150	0	0
10	400	400	100	100	200	200	100	100	100	200	200	526	314	0	0
11	400	400	100	100	200	200	100	100	100	200	200	526	314	0	0
12	400	400	100	100	200	200	100	100	100	200	200	390	150	0	0
13	400	400	100	100	200	200	100	100	100	200	200	526	164	0	150
14	400	400	100	100	200	200	100	100	100	200	200	526	314	0	0
15	400	400	100	100	200	200	100	100	100	200	200	526	314	0	0
16	400	400	100	100	200	200	100	100	100	200	200	526	414	0	0
17	400	400	100	100	200	200	100	100	100	200	200	526	374	150	0
18	400	400	100	100	200	200	100	100	100	200	200	526	464	150	0
19	400	400	100	100	200	200	100	100	100	200	200	526	464	150	0
20	400	400	100	100	200	200	100	100	100	200	0	526	508	306	0
21	400	400	100	100	200	200	100	100	100	200	0	526	508	306	0
22	400	400	100	100	200	200	100	100	100	200	0	526	508	216	0
23	400	400	100	100	200	200	100	100	100	200	0	526	0	464	0
24	400	400	100	100	200	200	100	100	100	200	0	526	0	414	0

Sementara itu, biaya pembangkitan tiap jamnya disajikan pada Tabel 4.17 berikut.

Tabel 4.17 Biaya pembangkitan tiap jam penelitian

Jam Ke-	Biaya Pembangkitan (Rp)
1	568.547.938
2	579.203.826,5
3	556.391.628
4	585.182.048,2
5	585.182.048,.2
6	519.518.849,6
7	412.640.605,6
8	547.733.249,8
9	613.972.468,4
10	661.213.106,2
11	661.213.106,2
12	593.323.258,6
13	668.935.241
14	661.213.106,2
15	661.213.106,2
16	686.218.527,1
17	721.737.129,7
18	744.527.614,4
19	744.527.614,4
20	737.132.010,1
21	737.132.010,1
22	715.161.993,2
23	642.445.559,9
24	629.726.481,4

4.2 Pembahasan

Algoritma genetika menghasilkan solusi yang konvergen ke nilai terbaik yang ditunjukkan pada Gambar 4.13. Seiring dengan meningkatnya jumlah iterasi, biaya pembangkitan yang diperoleh semakin menurun atau lebih kecil daripada nilai yang diperoleh pada beberapa iterasi sebelumnya. Hal ini dikarenakan dalam algoritma genetika terdapat proses reproduksi jadwal dengan tahapan seleksi, *crossover* dan mutasi sehingga jadwal yang diperoleh pada sebuah iterasi merupakan perbaikan dari jadwal yang diperoleh pada iterasi sebelumnya. Pada tahapan seleksi, jadwal dengan

nilai *fitness* rendah atau biaya pembangkitan yang besar akan terleminasi dan digantikan oleh jadwal dengan nilai *fitness* terbaik (biaya pembangkitan minimal) yang selanjutnya akan menjadi induk pada tahapan *crossover*. Induk terbaik ini disilangkan pada tahapan *crossover* dan menghasilkan anak terbaik. Tahapan mutasi menjaga agar tidak terjadi kerusakan jadwal (berubahnya jadwal menjadi tidak optimal) selama tahapan *crossover*. Oleh karena itu, algoritma genetika menghasilkan solusi yang lebih baik dari iterasi ke iterasi sebelumnya.

Grafik pada Gambar 4.13 juga menunjukkan biaya pembangkitan menurun tajam pada iterasi ke-1 sampai iterasi ke-225 (lihat lampiran C). Selanjutnya pada iterasi ke-227 sampai dengan iterasi ke-529 biaya pembangkitan menurun tetapi tidak setajam pada iterasi sebelumnya. Pada iterasi ke-529 hingga iterasi ke-920, biaya pembangkitan masih mengalami penurunan tetapi terlihat bahwa biaya pembangkitan semakin konvergen ke nilai terbaik. Setelah iterasi ke-920, biaya pembangkitan tidak lagi mengalami perubahan. Dengan demikian jadwal terbaik diperoleh pada iterasi ke-920 yang disajikan pada Tabel 4.15.

Dari Tabel 4.15 diperoleh hasil sebagai berikut. Unit 1, 2, 6, 7, 8, 9, dan 10 selalu beroperasi selama 24 jam. Unit 3, beroperasi mulai jam ke-7 (06.00 WIB) sampai jam ke-24 (24.00 WIB). Unit 4 beroperasi mulai jam ke-4 (03.00 WIB) sampai jam ke-24 (24.00 WIB). Unit 5 beroperasi mulai jam ke-2 (01.00) sampai jam ke-24 (24.00 WIB). Unit 11 beroperasi mulai jam ke-1 (00.00 WIB) sampai jam ke-19 (19.00 WIB). Unit 12 selalu beroperasi kecuali pada jam ke-6 (05.00 WIB) hingga jam ke-7 (07.00 WIB). Unit 13 selalu beroperasi kecuali pada jam ke-7 (06.00 WIB) hingga jam ke-8 (08.00 WIB) dan pada jam ke-23 (22.00 WIB) hingga jam ke-24 (24.00 WIB). Unit 14 beroperasi mulai jam ke-17 (16.00 WIB) hingga jam ke-24 (24.00 WIB). Unit 15 hanya beroperasi pada jam ke-15 (14.00 – 15.00 WIB).

Beban harian yang harus disuplai pada jam pertama dan kedua sama yakni sebesar 2640 MW akan tetapi jadwal dihasilkan berbeda. Perbedaan ini terjadi karena algoritma genetika membangkitkan kemungkinan solusi awal secara acak. Sehingga meskipun beban harian yang harus disuplai sama besar, sangat memungkinkan jadwal

yang dihasilkan berbeda. Perbedaan jadwal inipun mempengaruhi besarnya biaya pembangkitan. Berdasarkan Tabel 4.16 diketahui besar biaya pembangkitan pada jam pertama adalah Rp 568.547.938,- sementara pada jam kedua Rp 579.203.826,5-. Dari kedua jadwal tersebut terdapat selisih biaya pembangkitan sebesar Rp 10.655.888,52-

Akan tetapi, jadwal pada jam-jam berikutnya akan cenderung sama dengan jadwal pembangkitan pada jam sebelumnya. Hal ini dapat dilihat dari jadwal kesembilan dan kesepuluh dimana beban harian yang harus disuplai berbeda tetapi jadwal yang dihasilkan sama. Dengan demikian biaya pembangkitan dapat diminimalisir karena tidak perlu ada tambahan *startcost* untuk unit-unit yang tetap dioperasikan pada jam-jam berikutnya. Sehingga dari jadwal pembangkitan tersebut dapat diketahui adanya kemiripan jadwal pembangkitan dalam dua puluh empat jam dan perbedaan unit-unit yang tidak beroperasi dalam tiap jamnya sangat sedikit. Selain karena faktor *startcost*, hal ini juga dipengaruhi adanya perubahan beban harian setiap jam dalam sehari (dua puluh empat jam) dimana beban harian cenderung meningkat (dapat dilihat pada lampiran B).

Pada jam kesembilan dan kesepuluh dihasilkan jadwal pembangkitan yang sama akan tetapi nilai *fitness* atau biaya pembangkitan yang dihasilkan berbeda karena adanya perbedaan daya yang dibangkitkan oleh masing-masing unit yang beroperasi. Disinilah *fitness scaling* berperan dalam menentukan jadwal yang akan bertahan pada iterasi berikutnya dan menghasilkan jadwal dengan *fitness* terbaik (biaya pembangkitan minimal) sebagai solusi akhir.

BAB 5. KESIMPULAN DAN SARAN

5.1 Kesimpulan

Dari hasil dan pembahasan dapat dibuat kesimpulan sebagai berikut. Jadwal optimal dari 15 unit pembangkit yang beroperasi di PLN PJB II diperoleh pada iterasi ke-920 dengan biaya pembangkitan sebesar Rp 17.093.816.034 (tujuh belas milyar sembilan puluh tiga juta delapan ratus enam belas ribu tiga puluh empat rupiah).

5.2 Saran

Dalam skripsi ini teknik *fitness scaling* yang digunakan hanya *exponential scaling* sementara masih banyak jenis fitness scaling yang dapat digunakan dan diteliti kinerjanya. Selain itu, *fitness scaling* sebagai salah satu proses dari algoritma genetika dapat diaplikasikan pada persoalan lain yang dapat diselesaikan dengan algoritma genetika.

DAFTAR PUSTAKA

Dispankar, D. & Douglas, M. R. 1993. *Short Term Unit Commmitment Using Genetic Algorithm.* Technical Report No IKBS-16-93, 10 August.

Golberg, D. E. 1984. *Genetic Algorithms in Search, Optimization, and Machine Learning*. New York : Addison-Wesley Publishing Company, Inc.

Imaduddin. 2008. *Pasar Ketenagalistrikan dan Struktur Pasar.* http://imadudd1n.wordpress.com [22 Maret 2011].

Kusumadewi, S. 2003. *Artificial Intelligence (Teknik dan Aplikasinya)*. Jogjakarta : Graha Ilmu.

Ladd, S. R. 1996. *Genetic Algorithms in C++*. New York : M&T Books

Marsudi, D. 2006. *Operasi Sistem Tenaga Listrik*. Jakarta: Balai Penerbit & Humas ISTN.

Sanjoyo. 2006. *Aplikasi Algoritma Genetika.* http://sanjoyo55.files.wordpress.com/2008/11/non-linier-gen-algol.pdf [17 Mei 2010].

Suyanto. 2005. *Algoritma Genetika dalam Matlab.* Jogjakarta : Penerbit Andi.

Wibowo, A. P. 2003. "Optimasi Biaya pada Penjadwalan Pembangkit Menggunakan Metode Pemrograman Dinamis Fuzzy." Tidak Diterbitkan. Skripsi. Surabaya : Institut Teknologi Sepuluh Nopember.

Wood, J. A. & Wollenberg, B. F. 1996. *Power Generation, Operation, and Control*. New York: John Wiley & Sons, Inc.

Lampiran A. DATA UNIT PEMBANGKIT DI PLN PJB II

No.	Nama Pembangkit	P_{mak} (MW)	P_{min} (MW)	*Startcost* (juta Rupiah)	Bahan Bakar	Harga Bahan Bakar (Rp/Mkal)	H
1.	PLTU Paiton 1	400	225	344,94	Batu Bara	42,589	$0.2P^2 + 2467{,}4P + 6000$
2.	PLTU Paiton 2	400	225	344,94	Batu Bara	42,589	$0.2P^2 + 2467{,}4P + 6000$
3.	PLTU Gresik 1	100	43	72,6	Gas	96,1636	$1.0P^2 + 2781{,}4P + 11250$
4.	PLTU Gresik 2	100	43	72,6	Gas	96,1636	$1.0P^2 + 2781{,}4P + 11250$
5.	PLTU Gresik 3	200	66	115,91	Gas	96,1636	$0{,}01P^2 + 2496{,}1P + 125250$
6.	PLTU Gresik 4	200	66	115,91	Gas	96,1636	$0{,}01P^2 + 2496{,}1P + 125250$
7.	PLTU Muara Karang 1	100	44	61,91	MFO	37,1054	$0{,}5P^2 + 3073{,}5P + 28125$
8.	PLTU Muara Karang 2	100	44	61,91	MFO	37,1054	$0{,}5P^2 + 3073{,}5P + 28125$
9.	PLTU Muara Karang 3	100	44	61,91	MFO	37,1054	$0{,}5P^2 + 3073{,}5P + 28125$
10.	PLTU Muara Karang 4	200	70	108,76	Gas	93,12285	$0{,}1P^2 + 2728{,}2P + 57000$
11.	PLTU Muara Karang 5	200	70	108,76	Gas	93,12285	$0{,}1P^2 + 2728{,}2P + 57000$
12.	PLTGU Gresik 1	526	200	73,32	Gas	96,1636	$0{,}1P^2 + 2059{,}3P + 559687$
13.	PLTGU Gresik 2	526	200	73,32	Gas	96,1636	$0{,}1P^2 + 2059{,}3P + 559687$
14.	PLTGU Gresik 3	526	200	73,32	Gas	96,1636	$0{,}1P^2 + 2059{,}3P + 559687$
15.	PLTGU Muara Karang	508	150	61,57	Gas	93,12285	$0{,}3P^2 + 2381{,}9P + 110053$

MKal = Mega kalori (satuan untuk energi dan kalor).

Lampiran B. BEBAN SISTEM PT PLN PJB II

Jam Ke-	Rentang Waktu	Beban Harian (MW)
1	00:00 – 01:00	2.640
2	01:00 – 02:00	2.640
3	02:00 – 03:00	2.540
4	03:00 – 04:00	2.640
5	04:00 – 05:00	2.640
6	05:00 – 06:00	2.490
7	06:00 – 07:00	2.090
8	07:00 – 08:00	2.490
9	08:00 – 09:00	2.740
10	09:00 – 10:00	2.940
11	10:00 – 11:00	2.940
12	11:00 – 12:00	2.640
13	12:00 – 13:00	2.940
14	13:00 – 14:00	2.940
15	14:00 – 15:00	2.940
16	15:00 – 16:00	3.040
17	16:00 – 17:00	3.150
18	17:00 – 18:00	3.240
19	18:00 – 19:00	3.240
20	19:00 – 20:00	3.240
21	20:00 – 21:00	3.240
22	21:00 – 22:00	3.150
23	22:00 – 23:00	2.890
24	23:00 – 24:00	2.840

Lampiran C. PENURUNAN BIAYA PEMBANGKITAN

No.	Iterasi Ke-	Banyak Iterasi	Biaya Pembangkitan	Selisih
1	1	1	19.903.521.075,4428	
2	2	1	19.615.729.281,2677	287.791.794,18
3	3	1	19.579.687.948,0211	36.041.333,25
4	4	1	19.307.965.276,6547	271.722.671,37
5	5	1	19.139.113.220,8922	168.852.055,76
6	6 - 7	2	19.122.173.199,0444	16.940.021,85
7	8 - 12	5	18.976.386.829,8784	145.786.369,17
8	13 - 19	7	18.968.245.619,5024	8.141.210,38
9	20	1	18.867.339.600,6714	100.906.018,83
10	21 - 22	2	18.794.178.062,5537	73.161.538,12
11	23	1	18.793.057.640,8284	1.120.421,73
12	24 - 28	5	18.617.457.443,8611	175.600.196,97
13	29 - 30	2	18.584.835.129,5079	32.622.314,35
14	31 - 49	19	18.433.615.615,3011	151.219.514,21
15	50 - 76	27	18.349.337.269,3327	84.278.345,97
16	77 - 80	4	18.227.638.852,4187	121.698.416,91
17	81 - 95	15	18.224.200.042,0827	3.438.810,34
18	96 - 97	2	18.220.003.685,3107	4.196.356,77
19	98 - 116	19	18.130.102.562,2127	89.901.123,10
20	117 - 119	3	18.120.295.920,9643	9.806.641,25
21	120 - 126	7	18.091.994.562,3139	28.301.358,65
22	127 - 168	42	18.004.326.397,8589	87.668.164,46
23	168 - 190	23	17.963.456.916,1791	40.869.481,68
24	191 - 201	11	17.950.607.805,2052	12.849.110,97
25	202 - 222	21	17.937.797.339,3462	12.810.465,86
26	223 - 224	2	17.929.656.128,9702	8.141.210,38
27	225 - 226	2	17.690.739.065,5470	238.917.063,42
28	227 - 242	16	17.682.597.855,1710	8.141.210,38
29	243 - 246	4	17.610.881.213,9227	71.716.641,25
30	247 - 268	22	17.559.732.161,5688	51.149.052,35
31	269 - 304	36	17.272.852.560,3662	286.879.601,20
32	305 - 333	29	17.229.043.188,2360	43.809.372,13
33	334 - 389	56	17.220.709.650,6600	8.333.537,58

No.	Iterasi Ke-	Banyak Iterasi	Biaya Pembangkitan	Selisih
34	390 - 486	97	17.214.134.514,1475	6.575.136,51
35	487 - 520	34	17.154.006.097,2335	60.128.416,91
36	521 - 528	8	17.150.221.097,9375	3.784.999,30
37	529 - 596	68	17.137.371.986,9636	12.849.110,97
38	597 - 603	7	17.134.684.283,2138	2.687.703,75
39	604 - 766	163	17.126.818.773,8790	7.865.509,33
40	766 - 773	8	17.118.677.563,5030	8.141.210,38
41	774 - 804	31	17.112.699.341,7491	5.978.221,75
42	805 - 853	49	17.104.558.131,3731	8.141.210,38
43	854 - 919	66	17.097.162.527,0254	7.395.604,35
44	920 - 999	80	17.093.816.033,7454	3.346.493,28

Lampiran D. *Script* Program Aplikasi Penjadwalan Operasi Sistem Pembangkit Tenaga Listrik dengan Algoritma Genetika

```
#include <stdlib.h>
#include <iostream.h>
#include <iomanip.h>
#include <stdio.h>
#include <conio.h>
#include <time.h>
#include <math.h>
#pragma hdrstop

FILE* WindScale;

const int POPSIZE = 20;
const int TIME = 24;
const int NOGEN = 15;
const int MAXGENS = 1000;
const double PMUTATION = 0.0009;
const double PCROSSOVER = 0.85;

int** GT[POPSIZE];
double** DayaMak;
double** DayaMin;
double** Popt[POPSIZE];
double** eval;
double* fitness;
double* TransCost;
double* OFValue;
int* POPBEST;
double* OFValueMin;
double** PoptBestOld;
double** PoptBestNow;
int I;
double Pmin[NOGEN]={

Pmin[0]=225,  Pmin[1]=225,  Pmin[2]=43,         Pmin[3]=43,
Pmin[4]=66,         Pmin[5]=66,   Pmin[6]=44,         Pmin[7]=44,
Pmin[8]=44,   Pmin[9]=70,         Pmin[10]=70,  Pmin[11]=200,
Pmin[12]=150,       Pmin[13]=150, Pmin[14]=150};

double Pmak[NOGEN]={
Pmak[0]=400,        Pmak[1]=400,        Pmak[2]=100,        Pmak[3]=100,
Pmak[4]=200,  Pmak[5]=200,        Pmak[6]=100,        Pmak[7]=100,
Pmak[8]=100,  Pmak[9]=200,              Pmak[10]=200, Pmak[11]=526,
Pmak[12]=508, Pmak[13]=508, Pmak[14]=508};

double PB[TIME]={
   PB[0]=2640,      PB[1]=2640,         PB[2]=2540,   PB[3]=2640,
       PB[4]=2640,
   PB[5]=2490,      PB[6]=2090,   PB[7]=2490,   PB[8]=2740,   PB[9]=2940,
   PB[10]=2940,     PB[11]=2640,  PB[12]=2940,  PB[13]=2940,  PB[14]=2940,
   PB[15]=3040,     PB[16]=3150,  PB[17]=3240,  PB[18]=3240,  PB[19]=3240,
   PB[20]=3240,     PB[21]=3150,  PB[22]=2890,  PB[23]=2840};
```

```
void inisial(void);
void evaluasi(void);
void CetakInisial(void);
int GetGenAkhir(int,int);
void CetakGenAkhir(void);
void LimitDaya(void);
double GetLimitDayaMin(int i, int j);
double GetLimitDayaMak(int i, int j);
void CetakLimitDaya(void);
double DayaOptimumMin(int k);
double DayaOptimumMak(int k);
void BebanFeasible(void);
void CetakBebanFeasible(void);
double PowerCost(int i, int j, int k);
void EvalCost(void);
void ObjectFuncCost(void);
void WindowsScaling(void);
void CetakEvalCost(void);
void CetakFitnessCost(void);
void ProbabilityFitness(void);

void inisial(void)
{
       int i,j,k;

   for (i=0; i<POPSIZE; i++)
       {
         for(j=0; j < TIME; j++)
              {
                     for(k =0; k < NOGEN; k++)
                            GT[i][j][k]=random(2) ;
         }
      }
}


void CetakInisial(void)
{
       int i,j,k;

       fprintf(WindScale,"\n================================================="
);
   fprintf(WindScale,"\nInisial dari Generator");
   fprintf(WindScale,"\n=================================================");
   for (i=0; i<POPSIZE; i++)
       {
      fprintf(WindScale,"\nPopulasi ke-%d :",i);
      for(j=0; j < TIME; j++)
                     {
             fprintf(WindScale,"\n[jam ke-%2d]",j+1);
                           for (k=0; k<NOGEN; k++)
                     fprintf(WindScale,"%2d",GT[i][j][k]);
         }
         }
}
```

```
//Fungsi menentukan jumlah daya maksimum dan daya minimum tiap jam

void LimitDayaMin(void)
{
   int i,j,k;
   double PowerMin;

   DayaMin = new double* [POPSIZE];
   for(int i = 0; i < POPSIZE; i++)
       DayaMin[i] = new double [TIME];

   for (i = 0; i < POPSIZE; i++)
              {
              for (j= 0; j < TIME; j++)
                     {
          PowerMin = 0;
              for(k =0; k < NOGEN; k++)
                           {
              if (GT[i][j][k])
                           PowerMin = PowerMin + Pmin[k];
                           }
                     DayaMin[i][j]=PowerMin;
          }
       }
}

void LimitDayaMak(void)
{
   int i,j,k;
   double PowerMak;

   DayaMak = new double* [POPSIZE];
   for(int i = 0; i < POPSIZE; i++)
       DayaMak[i] = new double [TIME];

   for (i = 0; i < POPSIZE; i++)
              {
              for (j= 0; j < TIME; j++)
                     {
          PowerMak = 0;
              for(k =0; k < NOGEN; k++)
                           {
              if (GT[i][j][k])
                           PowerMak = PowerMak + Pmak[k];
                           }
                     DayaMak[i][j]=PowerMak;
          }
       }
}

double DayaOptimumMin(int k)
{
   double POpt;

   POpt = Pmin[k];
   return POpt;
}
```

```
double DayaOptimumMak(int k)
{
   double POpt ;

   POpt = Pmak[k];
       return POpt;
}

void BebanFeasible(void)
{
   int i,j,k;
       double DayaCom, sisa;
   double PowerCom, POpt, PCom;
   /*
   //Beban tiap jamnya;
       double PB[TIME]={
   PB[0]=2640,      PB[1]=2640,         PB[2]=2540,   PB[3]=2640,
       PB[4]=2640,
   PB[5]=2490,      PB[6]=2090,   PB[7]=2490,   PB[8]=2740,   PB[9]=2940,
   PB[10]=2940,     PB[11]=2640,  PB[12]=2940,  PB[13]=2940,  PB[14]=2940,
   PB[15]=3040,     PB[16]=3150,  PB[17]=3240,  PB[18]=3240,  PB[19]=3240,
   PB[20]=3240,     PB[21]=3150,  PB[22]=2890,  PB[23]=2840};
   */

   //Pengalokasian Daya Optimum terbaik untuk daya optimum generasi
sebelumnya
       PoptBestOld = new double* [TIME];
   for(int j = 0; j < TIME;j++)
              PoptBestOld[j] = new double [NOGEN];


   //Daya optimum populasi terbaik untuk generasi sebelumnya;
   for(j = 0; j < TIME;j++)
       {
      for(k = 0; k < NOGEN ;k++)
                    PoptBestOld[j][k]=PoptBestNow[j][k];
      }

   for(i =0; i < POPSIZE; i++)
         {
              for(j =0; j < TIME; j++)
               {

                            if(PB[j] <= DayaMin[i][j])
            {
            for(k=0; k < NOGEN; k++)
             {
               if (GT[i][j][k])
                            Popt[i][j][k]=DayaOptimumMin(k);
               else
                    Popt[i][j][k]=0;
               }
            }



         if(PB[j] == DayaMak[i][j])
```

```
              {
             for(k=0; k < NOGEN; k++)
              {
                if (GT[i][j][k])
                           Popt[i][j][k]=DayaOptimumMak(k);
                else
                     Popt[i][j][k]=0;
                }
             }

if((PB[j] > DayaMin[i][j]) && (PB[j] < DayaMak[i][j]))
              {
             for(k=0; k < NOGEN; k++)
              {
                if (GT[i][j][k])
                     Popt[i][j][k] = Pmin[k];
                else
                     Popt[i][j][k] = 0;
                }

                DayaCom = DayaMin[i][j];
                sisa = PB[j] - DayaCom;
                k = 0;

                while(sisa != 0 )
                     {
                   if (GT[i][j][k])
                     {
                      POpt = Popt[i][j][k] + sisa;

                      if(POpt > Pmak[k])
                            {
                         POpt = Pmak[k];
                         }

                      DayaCom = DayaCom + POpt - Popt[i][j][k];
                      Popt[i][j][k] = POpt;
                      sisa = PB[j] - DayaCom;
                      }
                   k=k+1;
                   }//batas while
             }//batas if dayamin < PB < dayamak

          if(PB[j] > DayaMak[i][j])
              {

             PowerCom = DayaMak[i][j];
             k = 0;
             while(PowerCom < PB[j])
              {
                if(GT[i][j][k]==0)  //menghidupkan generator lain untuk
memenuhi beban
                   {
                GT[i][j][k]=1;  // dari off-line menjadi on-line
                PowerCom = PowerCom + Pmak[k];
                }
             k = k + 1;
             }
```

```
                        DayaMak[i][j]=PowerCom;

            PCom = 0;
            for(k=0; k < NOGEN; k++)
             {
               if (GT[i][j][k])
                  {
                    Popt[i][j][k] = Pmin[k];
                  PCom = PCom + Pmin[k];
                  }
               else
                                      Popt[i][j][k] = 0;
               }

               DayaMin[i][j] = PCom;
               sisa = PB[j] - PCom;
               k = 0;

               while(sisa != 0 )
                    {
                  if (GT[i][j][k])
                    {
                     POpt = Popt[i][j][k] + sisa;

                     if(POpt > Pmak[k])
                           {
                        POpt = Pmak[k];
                        }

                     PCom = PCom + POpt - Popt[i][j][k];
                     Popt[i][j][k] = POpt;
                     sisa = PB[j] - PCom;
                     }
                  k=k+1;
                  }//batas while

            }
        }
      }
        for(int i = 0; i < POPSIZE; i++)
        delete [] DayaMak[i];
   delete [] DayaMak;

   for(int i = 0; i < POPSIZE; i++)
       delete [] DayaMin[i];
   delete [] DayaMin;
}

//Objective Function Pada setiap Generator
double PowerCost(int i, int j, int k)
{
   double a[NOGEN]={
   a[0]=0.2, a[1]=0.2,    a[2]=1.0,          a[3]=1.0,
   a[4]=0.01,       a[5]=0.01,   a[6]=0.5,          a[7]=0.5,
   a[8]=0.5, a[9]=-0.1,          a[10]=0.1,               a[11]=0.1,
   a[12]=0.3,a[13]=0.3,   a[14]=0.3};
```

```
      double b[NOGEN]={
   b[0]=2467.8,     b[1]=2467.8,        b[2]=2781.4,        b[3]=2781.4,
   b[4]=2496.1,     b[5]=2496.1,        b[6]=3073.5,        b[7]=3073.5,
   b[8]=3073.5,     b[9]=2728.2,        b[10]=2728.2,       b[11]=2059.3,
   b[12]=2381.9,    b[13]=2381.9,    b[14]=2381.9     };

      double c[NOGEN]={
      c[0]=6000,          c[1]=6000,    c[2]=11250,  c[3]=11250,
   c[4]=125250,     c[5]=125250,  c[6]=28125,         c[7]=28125,
   c[8]=28125,      c[9]=57000,         c[10]=57000,        c[11]=559687,
   c[12]=110053,    c[13]=110053, c[14]=110053};

   //Biaya Bahan bakar uuntuk masing-masing unit dalam Rp/MWh
   double o[NOGEN]={
   o[0]=42.589,            o[1]=42.589,   o[2]=96.1636,
      o[3]=96.1636, o[4]=96.1636, o[5]=96.1636,
   o[6]=37.1054,           o[7]=37.1054,  o[8]=37.1054,
      o[9]=93.12285,             o[10]=93.12285,     o[11]=96.1636,
   o[12]=96.1636,   o[13]=96.1636, o[14]=93.12285};

   double cost;

      cost = (a[k]*Popt[i][j][k]*Popt[i][j][k] + b[k]*Popt[i][j][k] +
c[k])*o[k];
   return cost;
}

//Menentukan nilai Objective Function pada setiap kromosom dan setiap Jamnya
void EvalCost(void)
{
      int i,j,k;
      double val;


   eval = new double* [POPSIZE];
   for(int i =0; i < POPSIZE; i++)
      eval[i] = new double [TIME];


   for(i =0; i < POPSIZE; i++)
      {
            for(j =0; j < TIME; j++)
                  {
        val = 0;
        for(k =0; k < NOGEN; k++)
            {
           if (GT[i][j][k])
                  {
             val = val + PowerCost(i,j,k);
             }
           }
           eval[i][j] = val;
        }
     }
}

//Fungsi Start-Up Cost untuk masing-masing unit
void StartUpCost(void)
```

```
{
       int i,j,k;
   double Transisi,startcost;
   double* CostStart;
   CostStart = new double [NOGEN];
   TransCost = new double [POPSIZE];

   double start[NOGEN] = {
   start[0]=344940000,     start[1]=344940000,        start[2]=72600000,
       start[3]=72600000,  start[4]=115910000, start[5]=115910000,
       start[6]=61910000, start[7]=61910000,   start[8]=61910000,
   start[9]=108760000,     start[10]=108760000,       start[11]=73320000,
       start[12]=61570000,        start[13]=61570000,
       start[14]=61570000};

       for(i = 0; i < POPSIZE; i++)
       {
      k = 0;
      while(k < NOGEN)
       {
         if(GT[i][0][k]==1)
              startcost = start[k];
         else
                          startcost = 0;

         for(j = 1; j < TIME; j++)
              {
              startcost = startcost + (GT[i][j][k] * (1-GT[i][j-
1][k])*start[k]);
             }
         CostStart[k] = startcost;
         k++;
         }

      Transisi = 0;
      for(k = 0; k < NOGEN; k++)
         {
         Transisi = Transisi + CostStart[k];
         }
      TransCost[i] = Transisi;
      }

   delete [] CostStart;
}


//Menentukan nilai Objective Function pada setiap populasi
void ObjectFuncCost(void)
{
       int i,j;
       double fittest;
   OFValue = new double [POPSIZE];

   for(i =0; i < POPSIZE; i++)
       {
      fittest = 0;
              for(j =0; j < TIME; j++)
       {
```

```
         fittest = fittest + eval[i][j];
         }
      OFValue[i] = fittest + TransCost[i];
      }

   delete [] TransCost;
}


//Menentukan nilai terbesar dari Objective Function pada setiap populasi
void FitnessTest(void)
{
       int i,j,k;
   double Cmin,Cmak;
   fitness = new double [POPSIZE];

   //menentukan nilai maksimal dari nilai Objective Function yang ada
   Cmak = 0;
   for(i =0; i < POPSIZE; i++)
       {
      if (OFValue[i] > Cmak)
         {
       Cmak = OFValue[i];
         }
      }

   //menentukan nilai minimal  dari nilai Objective Function yang ada
   Cmin = 1000000000000.0;
   for(i=0; i < POPSIZE; i++)
       {
      if (OFValue[i] < Cmin)
         {
       Cmin = OFValue[i];
         POPBEST[I] = i;
         }
      }


       //Nilai populasi Objective Function terkecil pada setiap Generasinya;
   OFValueMin[I] = Cmin;

       //Daya optimum populasi terbaik
   for(j=0; j<TIME; j++)
             {
             for(k = 0; k < NOGEN; k++)
             PoptBestNow[j][k] = Popt[POPBEST[I]][j][k];
       }


   //Pengecekan apakah nilai OFValueMin[I] dengan OFValueMin[I-1] lebih
besar;
   if(OFValueMin[I] > OFValueMin[I-1])
             {
      OFValueMin[I] = OFValueMin[I-1];
      POPBEST[I] = POPBEST[I-1];
      for(j=0; j<TIME; j++)
                   {
                   for(k = 0; k < NOGEN; k++)
```

```
                    PoptBestNow[j][k] = PoptBestOld[j][k];
                }
      }

   fprintf(WindScale,"\n[GENERASI-%5d]  %15.4f",I,OFValueMin[I]);

       //Menentukan nilai fitness sekaligus merupakan exponential Scaling
   //fitness[i] = Cmak - OFValue[i]
   for(i=0; i < POPSIZE; i++)
       {
             fitness[i] = Cmak - OFValue[i];
      fitness[i] = (fitness[i] +1) * (fitness[i] +1);
      if(fitness[i]==0)
         fitness[i]=1;
      }

delete [] OFValue;

//Pendealokasian dari Daya Optimum terbaik untuk generasi sebelumnya
PoptBestOld
for(int j = 0; j < TIME; j++)
      delete [] PoptBestOld[j];
delete [] PoptBestOld;

}

//Menentukan reproduksi dengan Roulett Whell Peluang fitness
void Reproduksi(void)
{
      int i,j,k,l,pos;
   double sum, sumfit, cek;
   double* ProbSig;
      double* ProbFit;
   double* randfit;
   int* newpos;
   int** NewGT[POPSIZE];

   for(int i = 0; i < POPSIZE; i++)
             NewGT[i] = new int* [TIME];
   for(int i=0; i < POPSIZE; i++)
             for(int j=0; j < TIME; j++)
             NewGT[i][j] = new int [NOGEN];


   ProbSig = new double [POPSIZE];
   ProbFit = new double [POPSIZE];
   randfit = new double [POPSIZE];
      newpos = new int [POPSIZE];


   //Menentukan jumlah seluruh fitness yang ada
   sumfit = 0;
   for(i = 0; i < POPSIZE; i++)
       {
      sumfit = sumfit + fitness[i];
      }

   //Peluang dari masing-masing fitness
```

```
for(i = 0; i < POPSIZE; i++)
    {
   ProbFit[i] = fitness[i]/sumfit;
          }

//Peluang masing-masing fitness sampai dengan sigma fitness/sumfit = 1;
sum = 0;
for(i = 0; i < POPSIZE; i++)
    {
   sum = sum + fitness[i];
          ProbSig[i] = sum/sumfit;
   }


//Probabilitas reproduksi untuk tiap populasi
    for(l = 0; l < POPSIZE; l++)
    {
   randfit[l] = rand()%10000/10000.0;
   }


//Pengecekan probabilitas terhadap nilai fitness ProbSig;

    for(i = 0; i < POPSIZE; i++)
    {
   newpos[i]=0;
   cek = randfit[i];
   j=0;
   pos=0;
   while(cek > ProbSig[j])
    {
      pos=j+1;
      j++;
      }
   newpos[i]=pos;
   }


//Menentukan Genotype baru;
for(i = 0; i < POPSIZE; i++)
    {
   for(j = 0; j < TIME; j++)
    {
      for(k = 0; k < NOGEN; k++)
           NewGT[i][j][k]=GT[newpos[i]][j][k];
    }
   }

//Genotype baru dimasukkan pada variable GT yang lama;
for(i = 0; i < POPSIZE; i++)
    {
   for(j = 0; j < TIME; j++)
    {
      for(k = 0; k < NOGEN; k++)
           GT[i][j][k]=NewGT[i][j][k];
    }
   }
```

```
      delete [] ProbSig;
   delete [] ProbFit;
   delete [] randfit;
      delete [] newpos;
   delete [] fitness;

   for(int i=0; i < POPSIZE; i++)
             for(int j=0; j < TIME; j++)
             delete NewGT[i][j];
   for(int i=0; i < POPSIZE; i++)
      delete NewGT[i];

}

//Genotype baru dengan memberlakukan CROSSOVER;
void Crossover(void)
{
      int i,j,k,m;
   double* randcross;
   int* popcross;
      int* timecross;
   int** GTCross[POPSIZE];
   int makcross;

for(int i = 0; i < POPSIZE; i++)
             GTCross[i] = new int* [TIME];
   for(int i=0; i < POPSIZE; i++)
             for(int j=0; j < TIME; j++)
             GTCross[i][j] = new int [NOGEN];

   popcross = new int[POPSIZE];
   timecross = new int[POPSIZE];
      randcross = new double[POPSIZE];

   //Menentukan nilai random untuk crossover

      for(m = 0; m < POPSIZE; m++)
      {
       randcross[m] = rand()%10000/10000.0;
      }

   //Pengecekan populasi Genotype GT yang memenuhi Crossover Probability
   //yaitu PCrossover = 0.78%;
   j=0;
      for(i = 0; i < POPSIZE; i++)
      {
       if(randcross[i] < PCROSSOVER)
          {
           popcross[j] = i;
           j++;
          }
      }
   makcross = j; //jumlah kromosom yang mengalami crossover

   //kromosom Crossover ada atau tidak sama dengan nol;
   if(makcross)
      {
      if(makcross%2!=0)
```

```
       {
         i=0;
         while(randcross[i] < PCROSSOVER)
              {
             i++;
             }

         //populasi ke-i akan menjadi populasi crossover terakhir
         popcross[makcross] = i;

         //fprintf(WindScale,"\nCrossover untuk tambahan = %2d
\n",popcross[makcross]);

         //menghasilkan makcross yang genap

              for(j = 0; j < ((makcross+1)/2); j++)
                     {
                     timecross[j] = rand()%24;
              }

         //Proses Crossover dilakukan secara berpasangan dengan berurutan;
         j=0;
         while(j < ((makcross+1)/2))
              {
              for(i = 0; i <= makcross; i=i+2)
                     {
              for(k = 0; k < NOGEN; k++)
                     {

       GTCross[popcross[i]][timecross[j]][k]=GT[popcross[i]][timecross[j]][k
];

       GTCross[popcross[i+1]][timecross[j]][k]=GT[popcross[i+1]][timecross[j
]][k];

                     //Proses penukaran dari kedua Populasi Crossover yang
dipilih;

       GT[popcross[i]][timecross[j]][k]=GTCross[popcross[i+1]][timecross[j]]
[k];

       GT[popcross[i+1]][timecross[j]][k]=GTCross[popcross[i]][timecross[j]]
[k];
                     }
                     j++;
              }
              }
         }


      //Jika Maksimum crossover adalah genap ;
      //Jumlah makcross harus genap agar yang dimutasikan bisa berpasangan;
      if(makcross%2==0)
              {
         //Menentukan posisi Crossover pada waktu TIME=24 jam yang diberikan
              for(j = 0; j < (makcross/2); j++)
                     {
                     timecross[j] = rand()%24;
```

```
             }


         //Proses Crossover dilakukan secara berpasangan dengan berurutan;
         j=0;
         while(j < makcross/2)
              {
            for(i = 0; i < makcross; i=i+2)
                  {
              for(k = 0; k < NOGEN; k++)
                  {

       GTCross[popcross[i]][timecross[j]][k]=GT[popcross[i]][timecross[j]][k
];

       GTCross[popcross[i+1]][timecross[j]][k]=GT[popcross[i+1]][timecross[j
]][k];

                  //Proses penukaran dari kedua Populasi Crossover yang
dipilih;

       GT[popcross[i]][timecross[j]][k]=GTCross[popcross[i+1]][timecross[j]]
[k];

       GT[popcross[i+1]][timecross[j]][k]=GTCross[popcross[i]][timecross[j]]
[k];
                  }
              j++;
              }
            }
       }
      }
   delete [] popcross;
   delete [] timecross;
   delete [] randcross;

   for(int i=0; i < POPSIZE; i++)
             for(int j=0; j < TIME; j++)
             delete GTCross[i][j];
   for(int i=0; i < POPSIZE; i++)
       delete GTCross[i];
}


void Mutation(void)
{
       int i,j,k;
   //int p;
   //int makmutate;
   double randmutate [POPSIZE][TIME][NOGEN];


       for(i = 0; i < POPSIZE; i++)
       {
      for(j = 0; j < TIME; j++)
       {
                  for(k = 0; k < NOGEN; k++)
            {
```

```
                    randmutate[i][j][k] = rand()%10000/10000.0;
              }
                    }
       }


   //Pengecekan populasi Genotype GT yang memenuhi Mutation Propability
   //yaitu PMUTATION  = 0.15%;
   //p=0;//menentukan jumlah bit yang mengalami mutasi
       for(i = 0; i < POPSIZE; i++)
       {
      for(j = 0; j < TIME; j++)
       {
                    for(k = 0; k < NOGEN; k++)
              {
                    if(randmutate[i][j][k] < PMUTATION)
                    {
                //Apabila Genotype bernilai satu maka diubah menjadi 0;
                //Demikian sebaliknya
                     if(GT[i][j][k]==0)
                     GT[i][j][k]=1;
                else
                                        GT[i][j][k]=0;

                //makmutate = p+1;
                //p++;
                     }
              }
       }
      }
}

void main(void)
{
       int j, k;
   /*
       double PB[TIME]={
   PB[0]=5649, PB[1]=5749, PB[2]=5891, PB[3]=5969, PB[4]=5681,
   PB[5]=5255, PB[6]=5387, PB[7]=5825, PB[8]=6027, PB[9]=6090,
   PB[10]=6022, PB[11]=5867, PB[12]=6006, PB[13]=6027, PB[14]=5985,
   PB[15]=6101, PB[16]=6447, PB[17]=6720, PB[18]=6790, PB[19]=6790,
   PB[20]=6610, PB[21]=6290, PB[22]=5922, PB[23]=5607};
   */

   POPBEST = new int [MAXGENS];
       OFValueMin = new double [MAXGENS];
       time_t t,first, second;

 //Pengalokasian Daya optimum yang sedang berlangsung;
       PoptBestNow = new double* [TIME];
   for(int j = 0; j < TIME;j++)
              PoptBestNow[j] = new double [NOGEN];

 //Pengalokasian Array untuk keadaan Pembangkit;
       for(int i = 0; i < POPSIZE; i++)
              GT[i] = new int* [TIME];
   for(int i=0; i < POPSIZE; i++)
              for(int j=0; j < TIME; j++)
```

```
            GT[i][j] = new int [NOGEN];

 //Pengalokasian Array untuk Daya Optimum;
      for(int i = 0; i < POPSIZE; i++)
            Popt[i] = new double* [TIME];
   for(int i=0; i < POPSIZE; i++)
            for(int j=0; j < TIME; j++)
            Popt[i][j] = new double [NOGEN];

   first = time(NULL);  /* Gets system time */

WindScale = fopen("WindScale.txt","w");

fprintf(WindScale,"\n/POPSIZE = %d",POPSIZE);
fprintf(WindScale,"\n/TIME = %d",TIME);
fprintf(WindScale,"\n/NOGEN = %d",NOGEN);
fprintf(WindScale,"\n/MAXGENS = %d",MAXGENS);
fprintf(WindScale,"\n/PMUTATION = %3.4f",PMUTATION);
fprintf(WindScale,"\n/PCROSSOVER = %3.4f",PCROSSOVER);

printf("\n/POPSIZE = %d",POPSIZE);
printf("\n/TIME = %d",TIME);
printf("\n/NOGEN = %d",NOGEN);
printf("\n/MAXGENS = %d",MAXGENS);
printf("\n/PMUTATION = %3.4f",PMUTATION);
printf("\n/PCROSSOVER = %3.4f",PCROSSOVER);

inisial();
//CetakInisial();

//Nilai awal dari OFValueMin[0]
OFValueMin[0]=10000000000000000.0;
//Nilai awal untuk
for(j = 0; j < TIME;j++)
      {
      for(k = 0; k < NOGEN ;k++)
                  PoptBestNow[j][k] = 0;
      }

I=1;
fprintf(WindScale,"\n====  Generasi ke :  Nilai Optimumnya =  =====");
printf("\n====  Generasi ke :  Nilai Optimumnya =  =====");

srand((unsigned) time(&t));
while(I < MAXGENS)
   {
       LimitDayaMin();
       LimitDayaMak();
       BebanFeasible();
       EvalCost();
   StartUpCost();
       ObjectFuncCost();
       FitnessTest();
       Reproduksi();
       Crossover();
       Mutation();
   I++;
   //printf("\nGenerasi ke-%3d adalah = %15.4f",I-1,OFValueMin[I-1]);
```

```
   }


fprintf(WindScale,"\nNilai minimum sampai generasi ke-%3d adalah =
%15.4f",I-1,OFValueMin[I-1]);
printf("\nNilai minimum sampai generasi ke-%3d adalah = %15.4f",I-
1,OFValueMin[I-1]);

// Generator On-line terbaik
fprintf(WindScale,"\nGenerator On Line terbaik sampai generasi ke-%3d\n",I-
1);
printf("\nGenerator On Line terbaik sampai generasi ke-%3d\n",I-1);

for(j=0; j<TIME; j++)
       {
   fprintf(WindScale,"[Kromosom Terbaik Time-%2d] : ",j+1);
   //printf("[Kromosom Terbaik Time-%2d] : ",j+1);

       for(k = 0; k < NOGEN; k++)
      {
      if(PoptBestNow[j][k])
                  {
         GT[POPBEST[I-1]][j][k]=1;
         fprintf(WindScale,"%2d",GT[POPBEST[I-1]][j][k]);
         }
             else
       {
         GT[POPBEST[I-1]][j][k]=0;
         fprintf(WindScale,"%2d",GT[POPBEST[I-1]][j][k]);
         //printf("%2d",GT[POPBEST[I-1]][j][k]);
         }
             }
   fprintf(WindScale,"\n");
   //printf("\n");
   }

//Pembebanan ekonomis terbaik
fprintf(WindScale,"\nPembebanan ekonomis Generator On Line terbaik sampai
generasi ke-%3d\n",I);
printf("\nPembebanan ekonomis Generator On Line terbaik sampai generasi ke-
%3d\n",I);

for(j=0; j<TIME; j++)
       {
   fprintf(WindScale,"[Time-%2d, PB=%g ]: ",j+1,PB[j]);
   printf("[Time-%2d, PB=%g ]: ",j+1,PB[j]);

       for(k = 0; k < NOGEN; k++){
       fprintf(WindScale," %g",PoptBestNow[j][k]);
      printf(" %g",PoptBestNow[j][k]);
   }

   fprintf(WindScale,"\n");
   printf("\n");

   }

//Biaya perjam-nya adalah sebagai berikut
```

```
fprintf(WindScale,"\nCost Setiap Jam-nya\n");
for(j=0; j<TIME; j++)
      {
   fprintf(WindScale,"[Time-%2d], Cost = %15.3f\n",j+1,eval[POPBEST[I-
1]][j]);
   printf("[Time-%2d], Cost = %15.3f\n",j+1,eval[POPBEST[I-1]][j]);

   }

second = time(NULL); /* Gets system time again */
fprintf(WindScale,"=======================================\n");
fprintf(WindScale,"Waktu yang dihabiskan adalah : %f
\n",difftime(second,first));
fprintf(WindScale,"=======================================\n");

printf("=======================================\n");
printf("Waktu yang dihabiskan adalah : %f \n",difftime(second,first));
printf("=======================================\n");


   fclose(WindScale);
   printf("\nSuksess!!\n");
printf("AlhamduLillah!!\n");
getch();

delete [] OFValueMin;
delete [] POPBEST;

//dealokasi untuk array eval
for(int i =0; i < POPSIZE; i++)
      delete [] eval[i];
delete [] eval;

//Pendealokasian dari Daya Optimum terbaik PoptBestNow
for(int j = 0; j < TIME; j++)
      delete [] PoptBestNow[j];
delete [] PoptBestNow;

//Pendealokasian dari daya optimum
  for(int i=0; i < POPSIZE; i++)
             for(int j=0; j < TIME; j++)
             delete Popt[i][j];
   for(int i=0; i < POPSIZE; i++)
       delete Popt[i];

//Pendealokasian dari keadaan Pembangkit
  for(int i=0; i < POPSIZE; i++)
             for(int j=0; j < TIME; j++)
             delete GT[i][j];
   for(int i=0; i < POPSIZE; i++)
       delete GT[i];

}
```