LAPORAN PENELITIAN

ALGORITMA GENETIK DALAM PEMROGRAMAN LINEAR

Oleh :

Drs. Putra Jaya, MT
(Ketua Tim Peneliti)

Penelitian ini dibiayai oleh :
Dana Rutin Universitas Negeri Padang
Tahun Anggaran 2000
Surat perjanjian kerja Nomor : 1498/K 12/KU/Rutin/2000
Tanggal 1 Mei 2000

UNIVERSITAS NEGERI PADANG
2000

ALGORITMA GENETIK DALAM PEMROGRAMAN LINEAR

PERSONALIA PENELITIAN

Ketua Peneliti : Drs. Putra Jaya, MT

Anggota : 1. Drs. Hasanuddin, MS

2. Drs. Aswardi, MT

ALGORITMA GENETIK DALAM PEMROGRAMAN LINEAR

Putra Jaya

ABSTRAK

Penelitian ini bertujuan menemukan nilai optimasi pada masalah pemrograman linear dengan menggunakan algoritma genetik. Kriteria pencapaian tujuan ditentukan oleh perbandingan beban komputasi yang dihasilkan antara algoritma genetik dengan metoda simpleks sebagai metoda bandingan. Semakin kecil beban komputasi yang dihasilkan, semakin berkualitas metoda yang digunakan.

Secara ideal, algoritma genetik digunakan untuk menyelesaikan masalah optimasi bersifat nonlinear. Dengan mengubah fungsi objektif menjadi bentuk fungsi evaluasi $f(x)$= *konstanta* + $f(x_1, x_2, \ldots, x_k)$ dan batasan kombinasi linear $A\underline{x} \leq b$ atau $A\underline{x} \geq b$ menjadi $\{x_1 = [\ell_1 \quad r_1], \cdots, x_k = [\ell_k \quad r_k]\}$ yang himpunan titik cembung $D = \prod_{k=1}^{q} \langle \ell_k, r_k \rangle$, algoritma genetik dapat digunakan untuk menemukan nilai optimasi pada masalah pemrograman linear. Area titik cembung menjamin semua jenis operator genetik khusus dalam sistem *Float Genetic Algorithm* (FGA) dapat digunakan secara bersama, sehingga perkembangan hasil operator yang mengarah menjadi batasan linear selama proses evolusi berlangsung dapat diantisipasi.

Hasil eksekusi simulasi pemrograman genetik memperlihatkan area titik cembung dengan jumlah variabel yang minimum pada kondisi titik terendah jumlah populasi dan generasi yang menghasilkan penyelesaian akhir, dapat ditemukan beban komputasi sesuai yang diharapkan. Setelah diuji dengan metoda simpleks, beban komputasi algoritma genetik lebih kecil dibandingkan metoda simpleks. Data ini dapat diinterpretasikan sebagai hasil penelitian bahwa unjukkerja algoritma genetik secara simulasi program lebih baik dibandingkan dengan metoda simpleks dalam kaitannya terhadap beban komputasi, *bila fungsi objektif dan batasan kombinasi linear pada masalah pemrograman linear telah diubah menjadi bentuk fungsi evaluasi dan ruang solusi sesuai dengan kebutuhan input algoritma genetik.*

PENGANTAR

Kegiatan penelitian merupakan bagian dari darma perguruan tinggi, di samping pendidikan dan pengabdian kepada masyarakat. Kegiatan penelitian ini harus dilaksanakan oleh Universitas Negeri Padang yang dikerjakan oleh staf akademikanya ataupun tenaga fungsional lainnya dalam rangka meningkatkan mutu pendidikan, melalui peningkatan mutu staf akademik, baik sebagai dosen maupun peneliti.

Kegiatan penelitian mendukung pengembangan ilmu serta terapannya. Dalam hal ini, Lembaga Penelitian Universitas Negeri Padang berusaha mendorong dosen untuk melakukan penelitian sebagai bagian yang tidak terpisahkan dari kegiatan mengajarnya, baik yang secara langsung dibiayai oleh dana Universitas Negeri Padang maupun dana dari sumber lain yang relevan atau bekerja sama dengan instansi terkait. Oleh karena itu, peningkatan mutu tenaga akademik peneliti dan hasil penelitiannya dilakukan sesuai dengan tingkatan serta kewenangan akademik peneliti.

Kami menyambut gembira usaha yang dilakukan peneliti untuk menjawab berbagai permasalahan pendidikan, baik yang bersifat interaksi berbagai faktor yang mempengaruhi praktek kependidikan, penguasaan materi bidang studi, ataupun proses pengajaran dalam kelas yang salah satunya muncul dalam kajian ini. Hasil penelitian seperti ini jelas menambah wawasan dan pemahaman kita tentang proses pendidikan. Walaupun hasil penelitian ini mungkin masih menunjukkan beberapa kelemahan, namun kami yakin hasilnya dapat dipakai sebagai bagian dari upaya peningkatan mutu pendidikan pada umumnya. Kami mengharapkan di masa yang akan datang semakin banyak penelitian yang hasilnya dapat langsung diterapkan dalam peningkatan dan pengembangan teori dan praktek kependidikan.

Hasil penelitian ini telah ditelaah oleh tim pereviu usul dan laporan penelitian Lembaga Penelitian Universitas Negeri Padang, yang dilakukan secara "blind reviewing'. Kemudian untuk tujuan diseminasi, hasil penelitian ini telah diseminarkan yang melibatkan dosen/tenaga peneliti Universitas Negeri Padang sesuai dengan fakultas peneliti. Mudah-mudahan penelitian ini bermanfaat bagi pengembangan ilmu pada umumnya, dan peningkatan mutu staf akademik Universitas Negeri Padang.

Pada kesempatan ini kami ingin mengucapkan terima kasih kepada berbagai pihak yang membantu terlaksananya penelitian ini, terutama kepada pimpinan lembaga terkait yang menjadi objek penelitian, responden yang menjadi sampel penelitian, tim pereviu Lembaga Penelitian dan dosen senior pada setiap fakultas di lingkungan Universitas Negeri Padang yang menjadi pembahas utama dalam seminar penelitian. Secara khusus kami menyampaikan terima kasih kepada Rektor Universitas Negeri Padang yang telah berkenan memberi bantuan pendanaan bagi penelitian ini. Kami yakin tanpa dedikasi dan kerjasama yang terjalin selama ini, penelitian ini tidak akan dapat diselesaikan sebagaimana yang diharapkan dan semoga kerjasama yang baik ini akan menjadi lebih baik lagi di masa yang akan datang.

Terima kasih.

Padang, Desember 2000
Ketua Lembaga Penelitian
Universitas Negeri Padang,

Prof. Drs. Kumaidi, MA., Ph.D.
NIP 130605231

DAFTAR ISI

Halaman

DAFTAR TABEL

Halaman

DAFTAR GAMBAR

Halaman

DAFTAR LAMPIRAN

Halaman

I. PENDAHULUAN

A. Latar Belakang Masalah

Pemakaian algoritma genetik untuk pemecahan masalah pencarian nilai optimasi bukan merupakan hal yang baru. Karya pemelopor John Holland dari Universitas Michigan Amerika Serikat pada tahun 1975 terbukti sebagai sebuah sumbangan yang berarti bagi penerapan ilmu pengetahuan dan rekayasa (*engineering*). Sejak itu hasil-hasil kerja penelitian dalam bidang ini dari dunia akademik dan industri tumbuh secara ekponensial. Perkembangan itu semakin pesat saat tersedianya komputer dengan biaya rendah memiliki kecepatan tinggi. Persoalan-persoalan rumit yang membutuhkan solusi simultan, dahulu dianggap sebagai masalah takterpecahkan dapat dipecahkan dengan algoritma genetik.

Algoritma genetik (GA) adalah suatu jenis struktur pencarian nilai optimal berdasarkan peniruan proses evolusi biologi. Algoritma ini memberikan suatu alternatif bagi proses penentuan nilai parameter dengan meniru cara reproduksi, pembentukan kromosom baru serta seleksi alami seperti yang terjadi pada makhluk hidup. Proses algoritma genetik dimulai dengan memandang sekumpulan parameter sebagai gen-gen dari sebuah kromosom. Masing-masing kromosom dievaluasi untuk kesegarannya dalam menyelesaikan tugas peniruan yang diberikan. Pada

setiap langkah waktu algoritma, kromosom yang paling tangguh diizinkan untuk reproduksi. Keturunan yang baru kemudian membentuk generasi berikutnya sebagai suatu organisme. Sekelompok organisme (kelompok kromosom yang disebut dengan isitilah *parent* = induk atau *mating pool*) membentuk himpunan yang disebut dengan populasi. Suatu populasi terdiri atas *string*, yang distrukturisasi oleh serangkaian nilai dalam bentuk bineri atau bilangan nyata. Nilai positif umumnya dikenal sebagai nilai *fitness* yang mewakili populasi sebagai satu solusi dalam domain sulusi. Nilai ini untuk setiap *string* dalam suatu populasi diperoleh melalui fungsi objektif berdasarkan permasalahan yang dihadapi. Semakin tinggi nilai *fitness* berdasarkan fungsi objektif yang telah ditentukan akan bertahan pada generasi berikutnya (Goldberg, 1989).

Berdasarkan seleksi alami, algoritma genetik bekerja untuk mencari struktur individu berkualitas tinggi yang terdapat dalam populasi. Proses pencarian nilai optimal dilakukan secara paralel dengan melaksanakan suatu prosedur iteratif untuk mengatur struktur individu dari suatu populasi sebagai kandidat solusi. Dengan menggunakan operator genetik, algoritma genetik melakukan proses seleksi dan rekombinasi antar individu dalam suatu siklus iterasi yang disebut dengan generasi. Elemen dasar yang diproses adalah *string* yang tersusun dari rangkaian *substring* (analogi: gen), masing-masing parameter telah diberi kode angka biner atau bilangan nyata yang terletak dalam batas ruang solusi. Pada tahap proses

seleksi dilakukan evaluasi terhadap kualitas setiap individu dalam populasi guna mendapatkan peringkat solusi. Dari hasil evaluasi dipilih secara acak individu-individu yang akan mengalami rekombinasi. Individu dengan kualitas yang lebih baik mempunyai kemungkinan yang lebih besar untuk dipilih sebagai calon-calon individu anggota populasi bagi generasi berikutnya. Pembentukan populasi baru dilakukan pada tahap rekombinasi dengan menerapkan operator genetik *crossover* dan mutasi secara acak terhadap calon-calon individu yang terpilih dalam tahap seleksi. Individu-individu baru yang diperoleh berbeda dengan individu sebelumnya. Siklus ini diulangi hingga suatu kriteria akhir yang diinginkan tercapai.

Algoritma genetik tidak lagi diperhitungkan sebagai algoritma tertuntun secara matematis (*mathematically guided algorithm*), seperti terjadi pada tipe gradien tradisional tentang prosedur optimisasi (*optimazing procedure*) dengan formulasi matematis yang sangat sulit. Pada kenyataannya, GA sangat berbeda dari gambaran tersebut. GA adalah sebuah even yang memiliki kebebasan dan fleksibilitas intrinsik untuk memilih solusi yang diinginkan menurut spesifikasi desain. Optima yang diperoleh merupakan produk akhir yang mengandung elemen-elemen terbaik hasil perkembangan generasi ke generasi dari generasi sebelumnya. Atribut individual yang lebih kuat cenderung dibawa ke generasi berikutnya. Rumusan permainannya adalah "*the fittest will win*" (yang tangguh akan tahan hidup) (Man dkk, 1996: 519).

Pemrograman linear (LP) adalah salah satu masalah optimasi berkendala yang paling sering digunakan, dan telah diterapkan secara luas dalam setiap bidang perencanaan produksi teknik industri, ekonomi, sosial, dan pemerintahan. Dalam penyelesaiannya, pemrograman linear menuntut fungsi objektif bernilai maksimum atau minimum bertanda positif dengan tunduk pada variabel pembatas (konstrain). Metoda efektif untuk penyelesaian masalah LP telah dikembangkan oleh George Danzit pada tahun 1947 yang disebut dengan metoda *Simpleks*. Metoda ini menuntut algoritma tertuntun secara matematis dan membutuhkan penghitungan yang berulang dengan metoda yang sama pada setiap iterasi sampai ditemukan suatu iterasi yang menyatakan proses telah mencapai konvergen.

Berdasarkan perkembangan basis pengetahuan di bidang algoritma genetik, melatarbelakangi penulis untuk meneliti penggunaan algoritma genetik sebagai suatu metoda untuk menyelesaikan masalah pemrograman linear. Metoda ini adalah suatu upaya menyelesaikan masalah pemrograman linear tanpa membutuhkan algoritma tertuntun secara matematik seperti yang terjadi pada metoda simpleks.

B. Identifikasi Masalah

Berdasarkan latar belakang masalah yang telah dikemukakan, penelitian ini mengkaji kemungkinan penerapan algoritma genetik untuk

menyelesaikan masalah optimasi pemrograman linear. Secara umum optimasi didefinisikan sebagai proses menemukan sasaran terbaik dari alternatif solusi masalah melalui proses analisis dengan meninjau beberapa kendala dan kombinasi kendala sesuai tujuan optimasi. Algoritma genetik sebagai algoritma pencarian heuristik (*heuristic search*) merupakan salah satu metoda pelacakan bagi optimasi yang memiliki *ciri-ciri tersendiri, stochastic, constrain, dan prosesnya nonlinear*. Pada metoda ini dimungkinkannya spekulasi pencarian baru berdasarkan informasi hasil proses sebelumnya dengan pemantauan tingkat kecermatan tertentu.

Pencarian dengan pemanfaatan sifat gabungan pengkodean kumpulan parameter, pelacakan kumpulan populasi, penggunaan informasi fungsi objektif dan dengan fasilitas transisi acak, membedakan teknik algoritma genetik dengan metoda biasa. Dalam algoritma genetik fungsi objektif beserta kendalanya sebagai pemodelan dalam melacak diubah menjadi fungsi evaluasi dan ruang solusi. Solusi diperoleh berdasarkan fungsi *fitness* berupa fungsi evaluasi yang ditentukan oleh ketepatan ruang solusi.

Selanjutnya algoritma genetik mempunyai beberapa kelebihan dibandingkan dengan algoritma lainnya. Diantaranya adalah kemampuan untuk tidak terjebak dalam minimum lokal. Proses pencarian nilai optimal secara bersama-sama terhadap suatu ruang solusi. Tidak membutuhkan

suatu gambaran yang lengkap dari masalah yang dipecahkan. Algoritma genetik hanya membutuhkan suatu fungsi objektif dalam bentuk fungsi evaluasi dari sistem, dan mampu menempatkan solusi mendekati optimal untuk masalah-masalah yang kompleks.

Dengan kelebihan yang dimiliki, algoritma genetik juga memiliki beberapa kelemahan seperti dikemukakan Man dkk (1996: 527) bahwa algoritma genetik tidak menjamin akan memberikan waktu respon dan beban komputasi jauh lebih kecil dibandingkan dengan metoda konvensional. Sifat dasar yang tidak menguntungkan ini membatasi ruang lingkup penggunaan algoritma genetik.

Uraian yang telah dikemukakan dipandang menarik dan perlu diteliti dalam penerapannya pada masalah pemrograman linear. Untuk itu masalah penelitian dapat diidentifikasi sebagai berikut :

1. bagaimana memodifikasi fungsi objektif beserta pertidaksamaan pembatas pemrograman linear yang bersifat linear menjadi bentuk fungsi evaluasi dan ruang solusi yang bersifat nonlinear, sehingga dapat digunakan sebagai input pemrograman genetik?,
2. bagaimana mengaplikasikan fungsi evaluasi dan ruang solusi, serta operator genetik secara terpadu agar diperoleh nilai optimasi yang memiliki beban komputasi relatif kecil dibandingkan metoda simpleks.

C. Pembatasan Masalah

Bertitik tolak pada identifikasi masalah, penelitian ini dibatasi pada upaya mengaplikasi algoritma genetik sebagai metoda penyelesaian pemrograman linear yang menghasilkan beban komputasi relatif kecil dibandingkan metode simpleks. Upaya-upaya tersebut dikelompokkan dalam dua bagian, yaitu input dan operator genetik pada proses algoritma genetik. Ditinjau dari segi input, permasalahan dibatasi pada strategi membentuk fungsi evaluasi yang mempunyai jumlah variabel minimal dengan memiliki jarak interval relatif sempit untuk masing-masing variabel ruang solusi. Pembatasan ini bertujuan agar proses penghitungan yang dilakukan komputer sebagai faktor penentu jumlah beban komputasi dapat diperkecil. Semakin minimal jumlah variabel dan semakin sempit jarak interval masing-masing variabel yang akan dilacak, semakin kecil beban komputasi yang dibutuhkan.

Dilihat dari segi operator algoritma genetik, faktor penentu memperkecil beban komputasi terdiri atas ukuran parameter genetik dan representasi solusi yang digunakan. Representasi solusi dapat dinyatakan dalam dua bentuk, yaitu *Binary Genetic Algoritm* (BGA) dan *Float Genetic algorithm* (FGA). Representasi solusi sistem FGA memiliki beban komputasi lebih kecil dibandingkan sistem BGA. Keunggulan FGA tidak terjadi konversi bilangan biner menjadi desimal, karena untaian kromoson dinyatakan dalam bentuk bilangan desimal. Ukuran paremeter operator

genetik dalam proses algoritma genetik akan menentukan siklus evolusi, peluang seleksi dan *crossover*, serta laju mutasi. Jumlah populasi dan generasi sebagai penentu siklus evolusi lebih dominan menentukan beban komputasi dibandingkan peluang seleksi, *crossover* dan laju mutasi. Namun ukuran populasi dan generasi tidak memiliki standar nyata. Semakin besar jumlah populasi dan generasi untuk setiap siklus proses evolusi semakin besar beban komputasi yang dihasilkan.

D. Perumusan Masalah

Berdasarkan pembatasan masalah yang telah dikemukakan, dapat dirumuskan permasalahan penelitian dalam bentuk kalimat pertanyaan sebagai berikut :

1. bagaimana cara merancang fungsi evaluasi, sehingga diperoleh jumlah variabel dan jarak interval masing-masing variabel ruang solusi yang dapat menghasilkan nilai optimasi pada beban komputasi relatif kecil dibandingkan metode simpleks,
2. Apakah representasi solusi sistem FGA sebagai salah satu operator genetik dapat menemukan nilai optimasi yang menghasilkan beban komputasi relatif kecil,
3. bagaimana strategi memperkecil jumlah populasi dan generasi untuk mempersingkat siklus proses evolusi yang menghasilkan nilai optimasi.

E. Tujuan Penelitian

Tujuan penelitian ini adalah dapat :

1. membuat model fungsi evaluasi bersama ruang solusi sebagai input pemrograman genetik sesuai kriteria.
2. menentukan representasi solusi yang sesuai untuk seluruh operator genetik sebagai upaya untuk memperkecil beban komputasi,
3. menentukan strategi untuk memperkecil jumlah populasi dan generasi, sehingga penghitungan komputasi dapat dipersingkatkan,

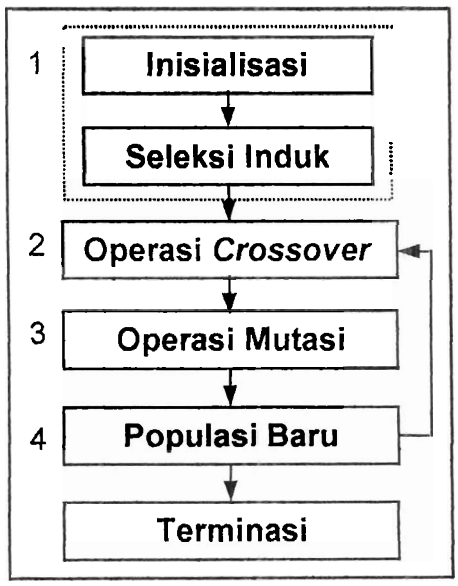
F. Manfaat Penelitian

Penelitian ini adalah suatu usaha untuk mendapatkan nilai optimasi pada masalah pemrograman linear dengan menerapkan metoda algoritma genetik. Keberhasilan penelitian diharapkan dapat memberikan hasil yang lebih efektif tanpa membutuhkan formulasi matematis yang rumit dalam menyelesaikan masalah pemrograman linear. Penelitian ini dapat memberi faedah dan manfaat dalam kemajuan dunia industri khususnya dan semua bidang yang berhubungan dengan masalah pemrograman linear.

II. TINJAUAN TEORI

A. Konsep Algoritma Genetik

Dalam algoritma genetik diberikan suatu alternatif bagi proses penentuan nilai parameter dengan meniru cara reproduksi genetik, pembentukan kromosom baru, serta seleksi alami seperti yang terjadi pada makhluk hidup. Secara umum proses evolusi algoritma genetik terdiri atas 4 tahap. Tentukan populasi awal secara acak. Lakukan evaluasi terhadap fungsi *fitness* berdasarkan tujuan yang ingin dicapai. Terapkan operasi genetik untuk mendapatkan alternatif solusi baru. Kemudian kembali ke tahap dua dan tiga seandainya maksimum generasi belum tercapai. Gambar 1 diperlihatkan proses evolusi algoritma genetik.



Gambar 1. Proses evolusi algoritma genetik

B. Operator Genetik

Untuk memperkenalkan *string* baru dalam populasi diperlukan operator genetik. Operator ini terdiri atas operator genetik dasar dan pengembangan. Operator genetik dasar merupakan bentuk yang paling sederhana, terdiri atas operator reproduksi, *crossover* dan operator mutasi.

1. Reproduksi

Reproduksi adalah proses pembentukan dan penyalinan nilai *fitness* untuk masing-masing *string* sebagai anggota populasi awal secara acak berdasarkan ruang solusi dan jumlah populasi. Selanjutnya setiap *string* dalam populasi akan mengalami seleksi berdasarkan nilai *fitness* untuk menentukan kesempatan menjadi induk. Seleksi individu untuk menghasilkan generasi yang sukses memainkan peranan penting dalam algoritma genetik. Individu dalam populasi dapat dipilih lebih dari satu kali di antara semua individu dalam populasi yang memiliki kesempatan untuk melakukan reproduksi pada generasi berikutnya. Besarnya peluang setiap individu dipilih sebagai induk ditentukan oleh nilai *fitness* yang dimiliki, sehinga individu yang lebih baik memiliki kesempatan untuk dipilih lebih banyak.

Pendekatan seleksi secara umum menunjukkan besarnya peluang seleksi P_j pada tiap individu j berdasarkan nilai *fitness*. Sederetan bilangan acak diberikan dan dibandingkan terhadap peluang komulatif $C_i = \sum_{j=1}^{i} P_j$ dari populasi yang bersangkutan. Nilai *Fitness* individu ke i dipilih dan disalin ke dalam populasi baru jika $C_{i-1} < U(0,1) \leq C_i$

Dalam pemrograman genetik paket program Matlab versi 4 dan 5 terdapat tiga jenis fungsi seleksi, yaitu jenis roda rolet (*roulette wheel*), ranking (*normgeomselect*) dan turnamen (*tournselect*). Metoda seleksi roda rolet adalah jenis fungsi seleksi tradisional dengan persamaan probabilitas *fitness* aktif ke *i* dibagi dengan jumlah *fitness* seluruh individual. Metoda seleksi turnamen yaitu seleksi berdasar kinerja turnamen. Algoritma genetik memilih dua individu secara acak, kemudian individu dengan nilai *fitness* tertinggi keluar sebagai pemenang. Metoda ini menerapkan pola perkawinan biologis, dua anggota populasi dari jenis kelamin yang sama bersaing untuk kawin dengan pihak ketiga yang berlainan kelamin. Metoda seleksi ranking yaitu seleksi berdasarkan peringkat yang ditetapkan dan bukan berdasarkan nilai numeris *fitness* suatu solusi dalam populasi.

Kinerja ketiga jenis fungsi seleksi ditentukan oleh faktor *bias*, *penyebaran* dan *efisiensi*. *Bias* didefinisikan sebagai perbedaan absolut antara probabilitas seleksi individual yang sebenarnya dengan probabilitas seleksi yang diharapkan. *Penyebaran* adalah daerah dalam jumlah ujicoba yang mungkin diperoleh individu. *Efisiensi* berkaitan dengan komplekstisitas waktu keseluruhan dari algoritma.

Sesuai dengan masalah dalam penelitian ini, digunakan seleksi jenis roda rolet. Metoda seleksi roda rolet merupakan cara seleksi pertama kali yang ditemukan Holand untuk memecahkan masalah

optimasi bersifat maksimasi. Metoda ini cendrung memiliki bias nol, berpotensi untuk menyebar secara tidak terbatas dengan komplekstisitas waktu berdasarkan $N \cdot \log N$. Peluang terpilihnya satu *string* untuk bereproduksi sebanding dengan nilai *fitness* yang dimiliki setiap *string* dibagi dengan nilai total *fitness* dalam populasi. Secara formula dinyatakan dengan persamaan (Hassoun, 1995: 440):

$$p_i = \frac{f(s_i)}{\sum_{j=1}^{M} f(s_j)} \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \quad (1)$$

p_i adalah besarnya peluang *string* i terpilih sebagai induk , $f(s_i)$ = nilai *fitness string* s_i dan $\sum_{j=1}^{M} f(s_j)$ adalah jumlah total *fitness* untuk seluruh *string* s_i dalam satu populasi. Semakin tinggi nilai p_i terhadap suatu *string*, semakin banyak *string* tersebut direproduksi. *String* dengan *fitness* lebih besar mempunyai kemungkinan direproduksi lebih banyak.

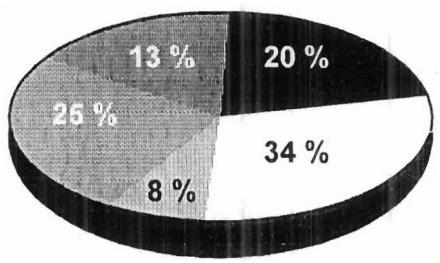
Tabel 1 memperlihatkan suatu populasi yang berhubungan dengan nilai *fitness* sebagai data singkat untuk lebih memperjelaskan prinsip metoda seleksi roda rolet (Robert , 1996: 12).

Tabel 1. Populasi yang berhubungan dengan nilai *fitness*

Kromosom	*Fitness*	Ruang Piringan
10110110	20	34%
10000000	5	8%
11101110	15	25%
10010011	8	13%
10100010	12	20%

Total *fitness* populasi tabel 1 adalah 60. Peluang terpilihnya kromosom 10110110 (kromosom urutan kesatu) untuk bereproduksi adalah (20:60).100% = 34%. Kromosom urutan kedua, tiga, empat dan lima secara berturut-turut memiliki peluang terpilihnya sebagai induk untuk menghasilkan kromosom baru adalah sebesar 8%, 25%, 13% dan 20%.

Proses seleksi jenis roda rolet dalam penerapannya diimplementasi dengan menggunakan piringan rolet. Masing-masing kromosom akan menempati sektor piringan rolet berdasarkan besarnya peluang terpilih sebagai induk, seperti diperlihatkan pada gambar 2.



Gambar 2. *Pie chart fitness* (Robert, 1996: 12)

Calon induk anggota populasi untuk generasi berikutnya, dipilih dengan memutar piringan rolet sebanyak jumlah *string*. Bila suatu putaran piringan rolet berhenti pada satu sektor, maka *string* yang menempati sektor tersebut dipilih sebagai calon induk anggota populasi baru. Peluang berhentinya piringan rolet pada suatu sektor sebanding dengan besarnya sektor, sehingga *string* dengan nilai *fitness* tertinggi mempunyai kemungkinan untuk dipilih menjadi induk anggota populasi pada generasi berikutnya.

Untuk menghasilkan kromosom baru, dipilih induk berdasarkan prosedur piringan rolet untuk menghasilkan 5 angka acak, seperti diperlihatkan pada tabel 2 (Robert, 1996: 13).

Tabel 2. Pemilihan kromosom induk dengan piringan rolet

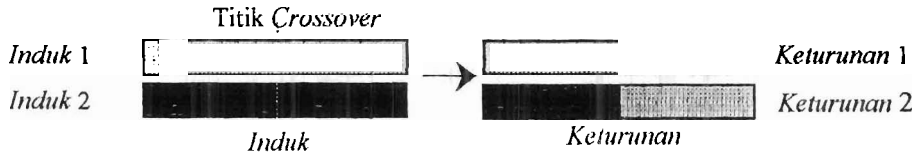
Kromosom	Bilangan acak terpilih
10010011	44
10110110	5
10100010	49
10110110	18
10000000	22

Kromosom 10110110 (terletak pada urutan ke 2 dan 4 dalam tabel 2) memiliki nilai *fitness* tertinggi terpilih menjadi induk 2 anggota dari populasi baru. Kromosom dengan nilai *fitness* lebih rendah (kecuali kromosom dengan nilai *fitness* terbaik kedua) hanya menjadi induk satu kali. Kromosom dengan nilai *fitness* terbaik kedua tidak bereproduksi. Hal ini menggambarkan suatu contoh pengacakan alamiah dari seleksi piringan rolet. Suatu *string* terpilih dibuat duplikasinya persis sama dengan *string* tersebut. Selanjutnya duplikasi *string* dikumpulkan dalam suatu *mating pool* untuk diproses lebih lanjut guna menghasilkan populasi baru.

2. Operator *crossover*

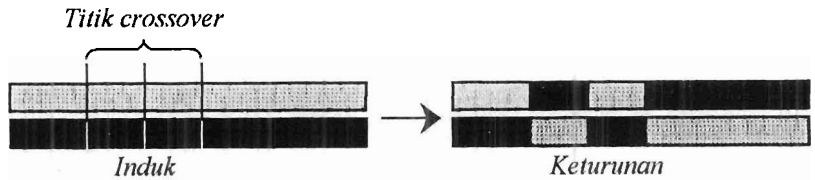
Dalam proses evolusi algoritma genetik berfungsi untuk menambah keanekaragaman *string* dalam populasi melalui penyilangan antar *string* yang diperoleh dari proses reproduksi sebelumnya. Proses

dimulai dengan memisahkan suatu *string* menjadi dua bagian. Selanjutnya salah satu bagian disilangkan dengan salah satu bagian dari *string* yang lain yang telah dipisahkan dengan cara yang sama. Proses ini dikenal dengan operator *crossover* satu titik, seperti diperlihatkan pada gambar 3 (Man dkk, 1996: 520).



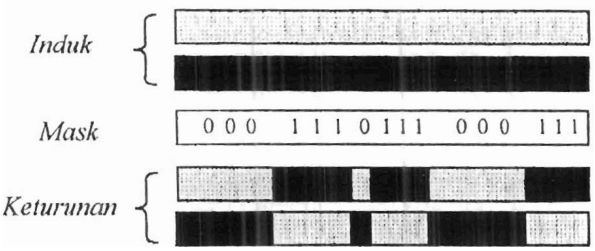
Gambar 3. Contoh *crossover* satu titik

Keturunan 1 (*offspring* 1) mengandung bagian depan induk 1 dan ekornya induk 2, sedangkan keturunan 2 mengandung bagian depan induk 2 dan ekornya induk 1. Sistem operator *crossover* satu titik diilhami proses biologis. Sistem ini memiliki kekurangan dalam beberapa situasi tertentu tidak dapat dikombinasikan. Untuk mengatasi masalah tersebut dibuat operator *crossover* banyak titik, seperti diperlihatkan pada gambar 4. *Crossover* banyak titik memperlihatkan kinerja kombinasi dapat dilakukan dalam beberapa situasi dengan menghasilkan keturunan dengan baik.



Gambar 4. Contoh *crossover* banyak titik, (m = 3)

Pendekatan lain untuk mengatasi kekurangan operator *crossover* satu titik adalah dengan menggunakan *crossover* seragam, seperti diperlihatkan pada gambar 5.



Gambar 5. Contoh *crossover* seragam

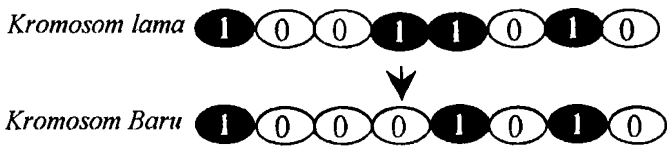
Dengan pendekatan *crossover* seragam dihasilkan keturunan yang mengandung campuran gen secara acak dari masing-masing induk.

3. Operator mutasi

Operator mutasi sering diperkenalkan sebagai suatu gambaran untuk menjaga konvergensi prematur (untuk solusi tidak optimal) yang disebabkan adanya informasi yang hilang. Biasanya operator reproduksi dan *crossover* bekerja dengan cukup efektif mencari dan melakukan rekombinasi terhadap *string* untuk mendapatkan *string* yang diinginkan. Namun kadang-kadang dapat terjadi kemungkinan terhapusnya suatu nilai yang penting dalam struktur tertentu dan tidak ada cara untuk memperoleh kembali nilai karakter yang hilang tersebut. Kehilangan nilai karakter disebabkan oleh *pool gen* cendrung menjadi lebih homogen bila satu gen mulai mendominasi. Untuk menjaga agar tidak

ada informasi yang hilang, melalui operator mutasi setiap *string* baru dapat diciptakan dengan melakukan modifikasi terhadap satu atau lebih nilai karakter pada *string* yang sama.

Tujuan mutasi mengatasi gangguan terhadap parameter untuk menjamin bahwa seluruh ruang lingkup dalam penelitian dapat dicapai. Umumnya, jika p_m lebih besar, rata-rata konvergensi lebih cepat tetapi menghasilkan suatu hasil *keadaan kesalahan yang sama* lebih besar. Dalam beberapa hal, analogi dengan parameter ukuran langkah (*stepsize*) dari algoritma gradien; ukuran langkah yang lebih besar mengakibatkan konvergensi lebih cepat tetapi kesalahan statemen pernyataan lebih tinggi, sedangkan ukuran langkah yang lebih kecil menyebabkan konvergensi menjadi lebih lambat dan kesalahan akhir lebih kecil. Proses mutasi secara acak mengubah gen dari 0 ke 1 atau dari 1 ke 0 dengan nilai tipikal probabilitas (p_m) kecil dari 0,1. Gambar 6 diperlihatkan proses mutasi untuk bit keempat (Man dkk, 1996: 520).



Gambar 6. Mutasi bit untuk bit ke empat.

4. Terminasi

Untuk mengoperasi algoritma genetik harus disediakan populasi awal melalui inisialisasi secara acak. Setelah proses evolusi

berlangsung, algoritma genetik dapat secara iteratif menyediakan populasi baru melalui hasil pengembangan proses mutasi. Populasi awal dapat diletakkan pada tempat urutan tertentu bersama penyelesaian yang cukup potensial dari populasi lainnya yang dihasilkan secara acak. Anggota populasi baru yang terbentuk merupakan gabungan individu terbaik hasil proses mutasi bersama anggota terbaik dari populasi lama. Terhadap anggota populasi baru dilakukan evaluasi untuk menentukan apakah siklus evolusi berlanjut atau berhenti dan melaporkan hasil.

Sebelum hasil penyelesaian ditemukan, algoritma genetik bergerak dari generasi ke generasi untuk menyeleksi dan menghasilkan induk hingga ditemukan kriteria akhir. Kriteria penghentian (teminasi) yang paling sering digunakan adalah berupa jumlah maksimum nilai *fitness* pada generasi tertentu. Strategi penghentian lainnya melibatkan kriteria konvergensi dengan memaksa keseluruhan populasi untuk berkumpul pada penyelesaian tunggal. Kriteria ini ditentukan oleh nilai standar deviasi terhadap nilai ambang tertentu. Bila nilai standar deviasi lebih kecil dibandingkan dengan nilai ambang yang ditetapkan, proses evolusi dapat dihentikan. Proses evolusi dapat dihentikan disebabkan kurangnya perbaikan dalam mendapatkan penyelesaian terbaik terhadap angka tertentu yang dibangkitkan setelah t kegagalan. Alternatifnya nilai target untuk pengukuran evaluasi dapat dicapai

berdasarkan pada sembarangan nilai ambang yang dapat dipakai. Ketiga kriteria penghentian dapat digunakan secara bersama satu sama lainnya.

C. Parameter Genetik

Ukuran parameter diperlukan untuk mengendalikan operator-operator genetik. Pemilihan ukuran parameter genetik menentukan penampilan kinerja algoritma genetik dalam memecahkan suatu masalah. Ukuran parameter yang sering digunakan terdiri atas ukuran populasi, probabilitas *crossover* (p_c) dan probabilitas mutasi (p_m).

Ukuran parameter populasi tidak memiliki standar nyata. Menurut Dejong dan Spears, J.J. Grefenstette dalam Man, dkk (1996: 520) semakin besar ukuran populasi, semakin cepat mencapai konvergensi dan akan membutuhkan komputasi lebih banyak dengan waktu yang lama untuk setiap populasi. Semakin besar nilai p_c semakin cepat struktur *string* baru diperkenalkan ke dalam populasi. Namun bila p_c terlalu besar, *string* sebagai calon pemecahan masalah terbaik mungkin akan hilang lebih cepat. Selanjutnya untuk mengendalikan laju mutasi diperlukan nilai parameter operator mutasi p_m lebih kecil dari p_c.

D. Optimasi Algoritma Genetik

Penerapan algoritma genetik untuk menentukan nilai optimasi didasarkan pada gambaran penyebaran titik yang mungkin dalam ruang

pencarian $D \subseteq R^q$, dimana $D = \prod_{k=1}^{q} \langle \ell_k, r_m \rangle$ dengan masing-masing variabel x_k dibatasi oleh interval $\langle \ell_k, r_k \rangle (1 \le k \le q)$. Dalam masalah optimasi terbatas, penyelesaian dengan ukuran satuan geometris dalam ruang R^q merupakan ciri yang paling menentukan. Sejumlah teori telah dikembangkan untuk mendapatkan bentuk himpunan penyelesaian optimasi dalam ruang R^q. Hanya ada satu jenis himpunan khusus untuk mendapatkan penyelesaian optimasi, yaitu himpunan titik cembung (*convex*). Misalnya masalah yang dihadapi adalah optimalkan $f(x_1, \cdots, x_q)$ dengan batasan $(x_1, \cdots, x_q) \in D \subseteq R^q$ dan D himpunan titik cembung. Domain himpunan titik cembung D terletak dalam batasan variabel $(\ell_k \le x_k \le r_k$ untuk $k = 1, \cdots, q)$ dan himpunan pembatas C. Dari kecembungan D, tiap titik pada area pencarian $(x_1, \cdots, x_q) \in D$ terdapat kemungkinan beberapa pembatas $\langle \ell(k), r(k) \rangle$ berdasarkan variabel $x_k (1 \le k \le q)$ dengan variabel lainnya $x_i (i = 1, \cdots, k-1, k+1, \cdots, q)$ tetap tidak berubah. Dengan kata lain, diberikan $(x_1, \cdots, x_k, \cdots, x_q) \in D$: $y \in \langle \ell(k), r(k) \rangle$ jika dan hanya jika $(x_1, \cdots x_{k-1}, y, x_{k+1}, \cdots . x_q) \in D$ dengan seluruh $x_i (i = 1, \cdots, k-1, k+1, \cdots, q)$ tetap konstan, dengan asumsi batasan $\langle \ell(k), r(k) \rangle$ dapat diperoleh secara efisien.

Masalah optimasi dalam domain cembung memiliki bentuk batasan terdiri atas :

a) Batasan domain : $\ell_i \leq x_i \leq u_i$ untuk $i = 1,2,\cdots,q$

b) Persamaan : $Ax = b$ dengan $x = \langle x_1,\cdots,x_q \rangle, A = (a_{ij}), b = \langle b_1,\cdots,b_p \rangle,$ $1 \leq i \leq p$ dan $1 \leq j \leq q$ (p adalah jumlah persamaan),

c) Pertidaksamaan: $Cx \leq d$ dengan $x = \langle x_1,\cdots,q \rangle, C = (c_{ij}), d = \langle d_1,\cdots d_m \rangle,$ $1 \leq i \leq m$ dan $1 \leq j \leq q$ (m adalah jumlah pertidaksamaan).

Untuk mendapatkan batasan yang tepat dari pembatas sebagai ruang pencarian GA, dapat dilakukan dengan teknik eliminasi dan substitusi terhadap persamaan pembatas. Cara lain adalah merancang operasi genetik khusus yang dapat menjamin semua kromosom terletak dalam area pencarian terbatas. Masalah ini dapat dicapai secara efisien pada pembatas berbentuk linear (Michalewicz, 1995: 122 -124).

1. Fungsi evaluasi dan ruang solusi

Fungsi *fitness* merupakan ukuran kinerja suatu individu agar tetap bertahan hidup dalam lingkungannya. Fungsi ini merupakan konsep yang paling penting dan sulit dalam perumusannya. Dalam algoritma genetik, fungsi *fitness* adalah fungsi evaluasi yang dibentuk oleh variabel ruang solusi dari masalah yang akan dioptimasi. Fungsi evaluasi yang diperoleh dari modifikasi fungsi objektif berfungsi sebagai ukuran *profit* yang ingin dimaksimalkan atau sebagai ukuran *cost* yang ingin diminimumkan. Untuk setiap masalah yang dioptimasi

membutuhkan pendefinisian fungsi *fitness* supaya *string* dengan kinerja yang tinggi dapat menghasilkan nilai fungsi *fitness* sesuai dengan tujuan.

Pemrograman linear (LP) adalah salah satu masalah optimasi berkendala. Rumusan persoalannya tetapkan vektor nonnegatif $\underline{x}$ dalam $\varphi = \underline{c}^T\underline{x}$ supaya φ bernilai optimal (maksimum atau minimum) bertanda positif dan $\underline{x}$ memenuhi $A\underline{x} \leq \underline{b}$ atau $A\underline{x} \geq \underline{b}$. Secara model persamaan ditulis dengan rumusan :

$$\text{optimumkan } \varphi = \underline{c}^T\underline{x} \quad \ldots\ldots\ldots\ldots\ldots\ldots \quad (2)$$

$$\text{dengan kendala : } A\underline{x} \leq \underline{b}$$

$$\text{atau } g_i(x) = A\underline{x} - b \leq 0$$

$$x \geq 0$$

$g_i(x)$ disebut sebagai fungsi kendala dan $x \geq 0$ disebut set kendala.

Pemodelan fungsi evaluasi dan ruang solusi untuk masalah pemrograman linear diperoleh dengan memodifikasi kombinasi variabel pembatas, sehingga bentuk pertidaksamaan pembatas kombinasi linear terurai menjadi persamaan variabel pembatas tunggal. Setiap variabel pembatas tunggal dibentuk menjadi batasan ruang solusi himpunan titik cembung, sehingga konstrain C menjadi himpunan kosong. Fungsi evaluasi diperoleh dengan mensubstitusi persamaan variabel pembatas tunggal ke dalam persamaan fungsi objektif pemrograman linear. Perubahan pertidaksamaan pembatas menjadi persamaan dilakukan

dengan menambah atau mengurangi setiap pertidaksamaan pembatas dengan variabel slack atau surplus variabel.

Anggap $Ax = b$ memiliki p persamaan independen yang terdiri dari p variabel $x_{i_1}, x_{i_2}, \cdots, x_{i_p} (\{i_1, \cdots i_p\} \subseteq \{1,2,\cdots,q\})$ yang dapat ditentukan oleh variabel lain. Untuk itu, matriks A dibagi secara vertikal menjadi larik A_1 dan A_2, sedemikian hingga kolom *ke* j pada matriks A termasuk dalam A_1 dan A_1 harus dapat dinvers:

$$Ax = b$$

$$[A_1 + A_2][X^1 X^2]^T = [b]$$

$$A_1 . X^1 + A_2 . X^2 = b$$

$$X^1 = A_1^{-1} \cdot b - A_1^{-1} \cdot A_2 \cdot X^2 \quad \ldots\ldots\ldots\ldots\ldots\ldots \quad (3)$$

Variabel X^1 adalah variabel pembatas tunggal terdiri dari $X^1 = (x_{i_1}, \cdots, x_{i_n})$ dan nilainya ditentukan oleh X^2. Variabel X^2 umumnya terdiri dari variabel slack atau surplus variabel. Nilai variabel X^2 dapat diperoleh melalui eliminasi dan substitusi nilai 0 dari seluruh variabel pembatas bersama nilai 0 dari sisa variabel slack atau surplus yang akan ditentukan nilainya ke dalam persamaan X^1. Kemudian nilai variabel pembatas tunggal sebagai ruang solusi diperoleh dengan mensubstitusikan nilai slack atau surplus variabel yang memenuhi ke dalam persamaan X^1. Untuk batas sisi yang lain variabel pembatas tunggal diperoleh dari batasan umum $x_i \geq 0$. Hasil modifikasi

pertidaksamaan pembatas kombinasi linear menjadi himpunan titik cembung sebagai ruang pelacakan bagi algoritma genetik dalam menemukan solusi optimal adalah $\{x_1 = [\ell_1 \quad r_1], \cdots, x_k = [\ell_k \quad r_k]\}$ dengan bentuk fungsi evaluasi $f(x)_{eval} = konstanta + f(x_1, x_2, \ldots, x_k)$.

Jumlah variabel ruang solusi pembentuk fungsi evaluasi mempengaruhi kinerja algoritma genetik. Semakin banyak nilai variabel yang akan dilacak semakin banyak algoritma genetik melakukan penghitungan, sehingga beban dan waktu komputasi menjadi besar. Oleh karena itu, dalam perancangan diarahkan supaya pemodelan fungsi evaluasi dibentuk dari variabel slack atau surplus variabel. Kedua jenis variabel ini pada umumnya memiliki jumlah lebih sedikit dibandingkan variabel utama dan kebanyakan bernilai nol.

2. *Operator genetik khusus*

Secara ideal optimasi dengan metoda algoritma genetik memiliki ciri tersendiri, yaitu proses evolusi bersifat nonlinear. Ruang pencarian penyelesaian terletak dalam himpunan titik cembung. Sifat ini menjadi kendala dalam penerapan algoritma genetik sebagai suatu metoda penyelesaian masalah pemrograman linear. Faktor penyebabnya adalah ruang pencarian algoritma genetik pada masalah pemrograman linear terletak pada daerah pertidaksamaan pembatas berbentuk linear. Sementara hasil operator akan berkembang dengan mudah menjadi batasan linear. Beberapa masalah yang timbul adalah :

a. ruang pencarian algoritma genetik pada masalah pemrograman linear sebagai penyelesaian semakin melebar, sehingga menimbulkan banyak alternatif solusi dan sulit mencapai optimum global;
b. proses evolusi sulit mencapai konvergen karena himpunan penyelesaian terletak dalam daerah linear (tidak cembung);
c. proses evolusi menjadi tidak stabil.

Masalah yang muncul dapat diatasi dengan membuat pembatas C menjadi himpunan kosong dan area pencarian $D = \prod_{k=1}^{q} \langle \ell_k, r_k \rangle$ menjadi himpunan titik cembung (Michalewicz, 1995: 123). Area cembung menjamin dua titik pencarian x membentuk kombinasi linear dengan menghasilkan satu titik penyelesaian dalam daerah D. Sifat tersebut memungkinkan dapat diterapkan operator *crossover* khusus meliputi *crossover* aritmetik, *crossover* sederhana dan operator *crossover* heuristik secara bersama dalam proses evolusi.

Setiap algoritma genetik membutuhkan representasi solusi sebagai gambaran tiap individu yang menjadi perhatian. Skema representasi menentukan bagaimana masalah disusun dalam algoritma genetik untuk menentukan operator yang akan digunakan. Tiap individu atau kromosom terbuat dari urutan gen yang dibentuk dari alfabet tertentu. Suatu alfabet dapat terdiri dari digit biner (0 dan 1), penyebaran nomor titik, integer, simbol (misalnya A, B, C, D) dan matriks. Dalam rancangan asli Holland, alfabet tersebut dibatasi pada digit biner. Oleh

karena itu, representasi dari suatu masalah telah menjadi subjek dari sekian banyak penyelidikan. Algoritma genetik telah menunjukkan bahwa representasi bersifat natural lebih efisien dalam menghasilkan solusi terbaik.

Suatu representasi solusi yang bermanfaat dari individu atau kromosom untuk optimasi fungsi adalah dengan melibatkan sebaran nilai gen atau variabel yang nilainya terletak dalam batas atas dan bawah ruang solusi. Pengkajian bidang ini telah diteliti oleh Michalewicz (1995:101-102) dengan melakukan perluasan percobaan melalui perbandingan kinerja sistem *Binary Genetic Algorithm* (BGA) dan *Float Genetic Algorithm* (FGA). Hasilnya menunjukkan bahwa FGA merupakan orde magnitud yang lebih efisien dalam kaitannya dengan beban dan waktu komputasi. Selanjutnya juga ditunjukkan bahwa FGA mendekati penyelesaian masalah dan menawarkan hasil dengan ketepatan yang lebih tinggi. Dalam penelitian ini akan diterapkan representasi solusi dengan menggunakan sistem FGA yang dibentuk oleh operator *crossover* sederhana, aritmetik, heuristik dan mutasi seragam serta *boundary* mutasi.

Operator *crossover* aritmetik dan *crossover* sederhana sangat diperlukan untuk mempercepat proses konvergensi dan menstabilkan proses evolusi. *Crossover* aritmetik dibentuk oleh kombinasi linear dari dua vektor. Jika x_1 dan x_2 disilangkan, keturunan yang dihasilkan

adalah $x_1^{'} = a.x_1 + (1-a).x_2$ dan $x_2^{'} = a.x_2 + (1-a).x_1$. Operator ini menggunakan nilai acak $a \in [0,1]$ dan $x_1^{'}$, $x_2^{'}$ berada dalam D. Nilai rerata *crossover* terjadi ketika a = 0,5. Selanjutnya operator *crossover* sederhana dibentuk dari $x_1 = (x_1, \cdots x_q)$ dan $x_2 = (y_1, \cdots, y_q)$. Jika keduanya disilangkan, keturunan yang dihasilkan adalah $x_1^{'} = (x_1, \cdots x_k, y_{k+1}, \cdots y_q)$ dan $x_2^{'} = (y_1, \cdots y_k, x_{k+1}, \cdots x_q)$. Operator ini dapat menghasilkan keturunan diluar ruang pencarian D. Untuk menghindari hal tersebut, digunakan area cembung yang terletak dalam batasan $a \in [0,1]$, sehingga :

$$x_1^{'} = \left\langle x_1, \cdots x_k, y_{k+1}.a + x_{k+1}(1-a), \cdots y_q.a + x_q.(1-a) \right\rangle$$

dan

$$x_2^{'} = \left\langle y_1, \cdots y_k, x_{k+1}.a + y_{k+1}(1-a), \cdots x_q.a + y_q.(1-a) \right\rangle$$

menjadi layak. Pertukaran informasi terbesar terjadi, jika nilai a terbesar. Metode yang paling sederhana dapat dimulai dengan a = 1, kemudian akan berkurang sebesar konstanta $1/\rho$ sampai a = 0. Keturunan akhir yang dihasilkan akan identik dengan induknya.

Crossover heuristik berfungsi untuk mendapat proses evolusi dengan menggunakan fungsi objektif dalam menentukan arah pencarian. *Crossover* ini menghasilkan keturunan tunggal dengan aturan $x_3 = a(x_2 - x_1) + x_2$, a adalah angka acak antara 0 dan 1. Untuk

kasus fungsi objektif maksimum, nilai $f(x_2) \geq f(x_1)$ dan minimum nilai $f(x_1) \geq f(x_2)$.

Selanjutnya konsep area cembung juga merupakan basis untuk dapat diterapkannya seluruh operator mutasi. Dengan demikian memungkinkan dapat diterapkan beberapa operator mutasi khusus secara bersama untuk mengatasi masalah yang timbul. Diantaranya adalah operator mutasi seragam dan mutasi *boundary*.

Mutasi seragam digunakan untuk memutasikan secara bebas gen-gen dalam ruang pencarian yang mungkin sebagai penyelesaian pada fase awal proses evolusi. Mutasi ini diperlukan pada kasus populasi awalnya terdiri dari salinan gen-gen berulang (jamak) dari titik yang sama (tunggal). Masalah tersebut sering terjadi pada optimasi terbatas (termasuk pemrograman linear), sehingga titik permulaan digunakan untuk proses yang sedang terjadi. Teknik mutasi seragam melakukan pengembangan iterasi dimulai dari titik tunggal. Proses iterasi selanjutnya akan dimulai pada titik terbaik berdasarkan proses iterasi sebelumnya. Teknik ini dapat mengatasi masalah batasan nonlinear pada daerah yang tidak terlalu cembung dan menemukan optimum global tanpa terjebak dalam optimum lokal. Dalam proses evolusi, mutasi seragam membutuhkan induk tunggal x dan menghasilkan keturunan tunggal x'. Operator memilih komponen acak $k \in (1,\ldots,q)$ dari vektor $x = (x_1,\ldots,x_k, \ldots,x_q)$ dan menghasilkan $x'=(x_1,\ldots,x_k',\ldots,x_q)$ dengan

$x_k^{'}$ merupakan nilai acak distribusi seragam terletak dalam batasan ruang pencarian $\langle \ell_k, r_k \rangle$.

Mutasi *boundary* merupakan variasi mutasi seragam dengan $x_k^{'}$ berada pada batas ℓ_k atau r_k, masing-masing mempunyai probabilitas yang sama. Operator mutasi *boundary* dalam proses evolusi digunakan untuk menemukan penyelesaian optimal yang terletak pada *boundary* atau didekat area pencarian yang dimungkinkan. Dalam proses evolusi algoritma genetik, mutasi *boundary* membutuhkan induk tunggal x untuk menghasilkan keturunan tunggal $x^{'}$.

Operator genetik sistem FGA secara formula dinyatakan dengan persamaan sebagai berikut (Houck dkk, tt : 4-6).

Crossover sederhana membangkitkan bilangan acak r distribusi seragam dari 1 sampai m dan menciptakan dua individu baru $\bar{x}^{'}$ dan $\bar{y}^{'}$ menurut persamaan :

$$x_i^{'} = \begin{cases} x_i \text{ , } jika\ i < r \\ y_i \text{ , } yang\ lain \end{cases}$$
$$y_i^{'} = \begin{cases} y_i \text{ , } jika\ i < r \\ x_i \text{ , } yang\ lain \end{cases} \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \quad (4)$$

Crossover aritmetik menghasilkan dua kombinasi linear komplementer pada induk (parent) dengan $r = U(0,1)$:

$$\bar{X}^{'} = r\bar{X} + (1-r)\bar{Y}$$
$$\bar{Y}^{'} = (1-r)\bar{X} + r\bar{Y} \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \quad (5)$$

Crossover heuristik membuat perhitungan linear dari dua individu. Operator ini merupakan satu-satunya operator yang memberikan sarana untuk memanfaatkan informasi *fitness*. Individu baru $\overline{X}'$ dibuat dengan menggunakan persamaan :

$$\begin{aligned} \overline{X}' &= r\overline{X} + r(\overline{X} - \overline{Y}) \\ \overline{Y}' &= \overline{X} \end{aligned} \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \quad (6)$$

dengan $r = U(0,1)$ dan $\overline{X}$ lebih baik dari $\overline{Y}$ dalam bentuk *fitness*. Jika $\overline{X}'$ tidak dimungkinkan yaitu kemungkinan sama dengan nol seperti dinyatakan oleh persamaan *feasibility* (kemungkinan terjadi) :

$$feasibility = \begin{cases} 1, & jika \quad x' \geq a_i, \quad x_i' \leq b_i \quad \forall i \\ 0, & yang \quad lain \end{cases} \quad \ldots\ldots\ldots\ldots\ldots\ldots \quad (7)$$

maka dibangkitkan bilangan acak baru r untuk menciptakan penyelesaian baru dengan mengunakan persamaan $\overline{X}' = r\overline{X} + r(\overline{X} - \overline{Y})$ atau berhenti. Untuk memastikan penghentian setelah t kegagalan, anggap anak sama dengan induk dan berhenti.

Mutasi seragam secara acak memilih satu variabel j dan mengatur variabel tersebut ke bilangan acak seragam $U(a_i, b_i)$ dengan mengunakan rumus :

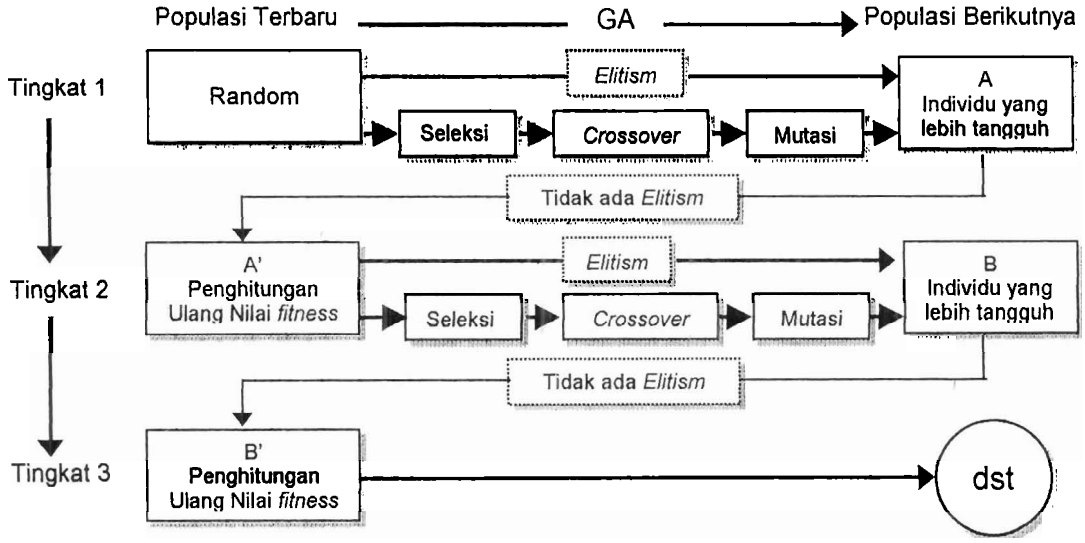
$$x_i' = \begin{cases} U(a_i, b_i), jika \quad i = j \\ x_i, \quad yang\ lain \end{cases} \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \quad (8)$$

Mutasi boundary secara acak memilih satu variabel j dan menempatkan sejajar dengan batas atas atau bawah ruang solusi dengan $r = U(0,1)$:

$$x_i' = \begin{cases} a_i, jika\ i = j,\ r < 0{,}5 \\ b_i, jika\ i = j,\ r \geq 0{,}5 \\ x_i, \qquad yang\ lain \end{cases} \quad \ldots\ldots\ldots\ldots\ldots\ldots\ (9)$$

3. Model elitism

Dalam setiap tingkat siklus proses evolusi algoritma genetik terjadi perubahan nilai *fitness*. Nilai *fitness* harus dihitung ulang pada setiap perpindahan tingkat. Mengingat fungsi *fitness* yang berbeda antar tahapan, diperlukan kajian yang tepat untuk mencegah individu-individu terbaik tidak hilang pada tingkat berikutnya. Untuk itu, perlu diterapkan metoda *elitism* untuk memelihara sejumlah anggota terbaik dari setiap populasi baru yang dihasilkan (Jang dkk, 1997). Metoda ini juga menerapkan dua individu terbaik yang dipelihara untuk melakukan persilangan dengan individu lain dalam satu generasi. Gambar 7 diperlihatkan urutan untuk mendapatkan dan memelihara individu-individu terbaik.



Gambar 7. Penerapan algoritma genetik dengan metode *elitism*

III. METODOLOGI DAN PELAKSANAAN PENELITIAN

A. Instrumentasi Penelitian

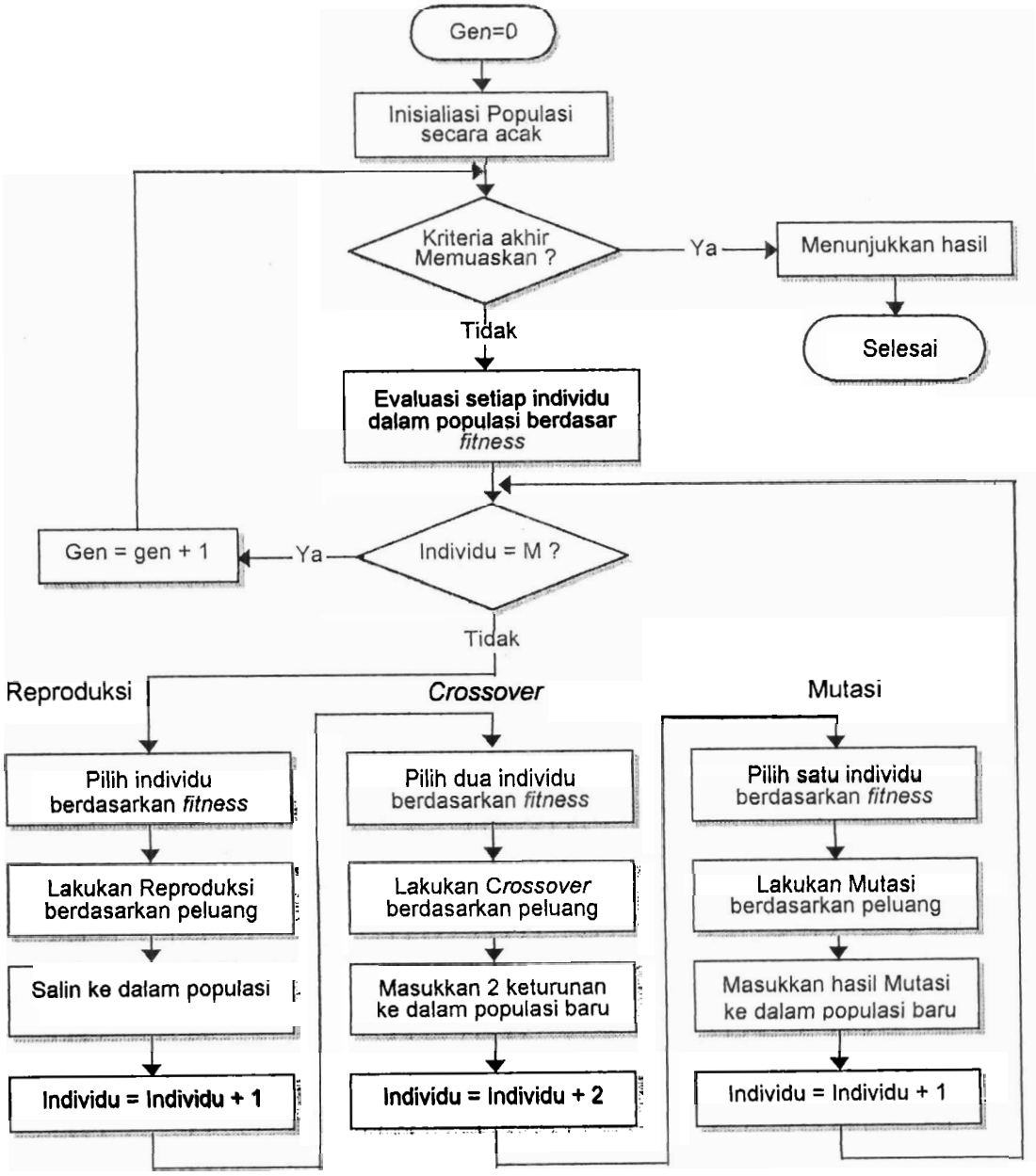
Data dalam penelitian ini diperoleh melalui hasil simulasi komputer. Oleh karena itu, proses evolusi algoritma genetik dilakukan dengan menggunakan program sebagai instrumentasi penelitian. Pemrograman genetik berbeda dengan algoritma genetik. Perbedaannya terletak pada representasi hasil yang digunakan. Pemrograman genetik menggunakan struktur program komputer dalam skema bahasa komputer sebagai solusi, sedangkan algoritma genetik menggunakan *string* sebagai mewakili solusi. Berdasarkan konsep-konsep yang telah dikemukakan dalam tinjauan teori, algoritma pemrograman genetik untuk masalah maksimasi nonnegatif dibentuk dengan langkah-langkah sebagai berikut (Jang dkk, 1997).

1. Inisialisasi suatu populasi dengan individu-individu yang dihasilkan secara acak dan evaluasi nilai *finess* masing-masing individu sesuai dengan kemampuan evaluasi yang dimiliki terhadap masalah yang ada;
2. (a). Pilih dua anggota populasi dengan probabilitas proporsional terhadap nilai-nilai *fitness*,
   (b). Terapkan *crossover* dengan probablitas yang sama dengan nilai *crossover*,
   (c). Terapkan mutasi dengan probabilitas yang sama dengan nilai mutasi,

(d). Ulangi (a) sampai (d) hingga anggota yang dihasilkan cukup untuk membentuk generasi berikutnya,

3. Ulangi langkah 2 dan 3 hingga kriteria berhenti ditemukan.

Diagram alir algoritma pemrograman genetik digambarkan pada gambar 8.



Gambar 8. Diagram alir pemrograman genetik.

Dalam penelitian ini, pemrograman genetik diimplementasikan dengan menggunakan paket program Matlab. Matlab merupakan lingkup teknis penghitungan numerik berpenampilan tinggi. Matlab menyatukan analisis numeris, komputasi matriks dan grafis dalam lingkup yang mudah untuk digunakan. Fungsi Matlab didefinisikan berupa file teks sederhana yang terdiri dari instruksi terinterpretasi. Fungsi Matlab sangat mudah dipindahkan dari suatu perangkat keras ke perangkat keras yang lainnya tanpa langkah rekompilasi.

Pemrograman genetik untuk keperluan penelitian ini, dimodifikasikan dari modul *Genetic Algorithm for Optimization Toolbox* (GAOT) paket program Matlab. GAOT merupakan sekelompok fungsi yang saling berhubungan. Fungsi dasar berupa fungsi *ga* yang berfungsi untuk menjalankan evolusi tersimulasi. Modul listing program terdapat pada lampiran A. Keperluan dasar fungsi *ga* diberikan oleh langkah-langkah perintah berikut :

$[x,endPop,bPop,traceInfo] = ga\ (bound,evalFN,evalParam,params,startPop,\ldots.$
$termFN,termParams,selectParams,xOverParams,mutFN,\ mutParams)$

Parameter output :

x adalah penyelesaian *string* terbaik, penyelesaian akhir,

$bPop$(**optional**) adalah matriks individu terbaik yang berhubungan dengan generasi tujuan yang ditemukan,

$traceInfo$(**optional**) adalah matriks nilai rata-rata dan maksimum fungsi dari populasi untuk masing-masing generasi,

Parameter input :

bounds adalah matriks batas atas dan bawah variabel,

evalFN adalah fungsi evaluasi dalam bentuk file m,

evalParams(**optional**) adalah sebuat matriks baris dari masing-masing parameter pada fungsi evaluasi yang ditetapkan pada [*NULL*],

params(**optional**) adalah sebuah vektor dari option, yaitu [*epsilon prob_param disp_param*] dimana *epsilon* berupa perubahan yang dibutuhkan untuk mempertimbangkan dua penyelesaian berbeda, *prob_param* 0 jika digunakan algoritma versi biner dan 1 bila mengunakan versi float, *disp_param* mengendalikan perkembangan algoritma, 1 menampilkan generasi terbaru dan nilai penyelesaian terbaik, 0 menghalangi keluaran selama program berjalan. Parameter ini diatur pada [$1e^{-6}$ 1 0],
startPop(**optional**) adalah matriks nilai penyelesaian masing-masing fungsi. Populasi awal diatur pada populasi yang tercipta secara acak melalui **inisialisai**,

termFN(**optional**) adalah nama fungsi terminasi yang diatur pada ['*maxGenTerm*'],

termParams(**optional**) adalah matriks baris dari parameter yang diatur pada [*100*],

selectFN(**optional**) adalah nama fungsi seleksi yang diatur pada ['*roulette wheel*']

selectParams(**optional**) adalah matriks baris dari parameter untuk fungsi seleksi yang diatur pada [*0,08*],

xOverFNs(**optional**) adalah bentuk *string* fungsi *crossover* yang terpisah yang ditetapkan pada ['*arithXover heuristicXover simpleXover*'] untuk versi float dan ['*simpleXover*'] untuk versi biner,

xOverParams(**optional**) adalah matriks parameter *crossover* yang ditetapkan pada [*2 0;2 3;2 0*]untuk versi float dan [*0,6*] untuk versi biner,

mutFNs(**optional**) adalah bentuk *string* operator mutasi yang diatur pada ['*boundaryMutation multiNonUnifMutation nonUnifMutation UnifMutation*'] untuk versi float dan ['*binaryMutation*'] untuk versi biner,
mutParams(**optional**) adalah matriks parameter mutasi yang ditetapkan pada [*4 0 0;6 100 3;4 100 3;4 0 0*] untuk versi float dan [*0,05*] untuk versi biner.

Algoritma genetik menampilkan evolusi tersimulasi dengan menggunakan *evalFN* untuk menentukan nilai *fitness* dari *string* penyelesaian. Operator *xOverFNs* dan *mutFNs* digunakan untuk mengubah *string* penyelesaian selama pencarian. Sistem ini mempertahankan modularitas dan fleksibilitas sebagai hasil keputusan untuk melewati seleksi, evaluasi dan fungsi terminasi sebagai daftar operator algoritma genetik. Selanjutnya operator genetik mampu menampilkan evolusi dengan menggunakan fungsi evaluasi, kombinasi seleksi, *crossover*, mutasi dan fungsi terminasi sesuai spesifikasi fungsi dan parameter yang berlaku.

Fungsi evaluasi merupakan penggerak algoritma genetik. Fungsi ini dipanggil oleh *ga* untuk menentukan *fitness* masing-masing *string* penyelesaian dalam ruang solusi yang dibangkitkan selama pencarian. Dalam penelitian yang dilakukan, fungsi evaluasi ditulis dalam file m dan diberi nama 'GaplinEval.m'. Listing program terdapat pada lampiran A. Fungsi pemanggil berupa format :

```
function [sol  val] = GaplinEval(sol,option)
```

Untuk menjalankan *ga* dengan menggunakan fungsi tes ini, dipakai salah satu fungsi pemanggil Matlab berikut :

```
bstX = ga([bounds],'GaplinEval')
bstX = ga([bounds],'persamaan fungsi evaluasi');
```

Fungsi pemanggil menggunakan semua parameter yang ditetapkan *ga* dan hanya menghasilkan penyelesaian terbaik yang ditemukan selama evolusi tersimulasi berjalan. Fungsi evaluasi mengambil parameter *sol* dan *val*. *Sol* adalah vektor baris dari elemen n+1. Elemen n pertama adalah parameter yang ditinjau. Elemen ke n+1 adalah nilai penyelesaian. Matriks parameter adalah matriks baris dari [*current_generation, evalParams*]. Fungsi evaluasi melapor kembali hasil nilai *sol* dan *val* sehingga dapat memperbaiki dan mengembangkan *string* bila diinginkan. Fungsi evaluasi adalah unik untuk optimasi masalah yang dihadapi. Setiap kali *ga* digunakan pada masalah yang berbeda, fungsi evaluasi harus dikembangkan guna menentukan *fitness* dari individu.

Fungsi-fungsi operator dalam pemrograman genetik digunakan untuk menciptakan penyelesaian baru berdasarkan penyelesaian-penyelesaian yang muncul dalam populasi. Terdapat dua jenis *fungsi* operator genetik, yaitu *crossover* dan mutasi. Fungsi *ga* memanggil masing-masing operator untuk menghasilkan penyelesaian baru.

Crossover mengambil dua individu dan menghasilkan dua individu baru. Fungsi pemanggil *crossover* adalah :

[c1, c2] = crossover(p1, p2, bounds, params)

p1 adalah induk pertama (parent) [*solution_string function_value*], *p2* adalah induk kedua, *bounds* adalah matriks batas atas dan bawah, *params* adalah matriks vektor [*current_generation, operatorParams*], *operatorParams* adalah parameter baris untuk operator *crossover* dan mutasi. Nilai pertama *operatorParams* adalah frekuensi dari penerapan operator. Untuk FGA, nilai ini adalah bilangan waktu diskrit untuk memanggil operator pada setiap generasi.

Mutasi merubah suatu individu menjadi satu penyelesaian baru. Fungsi pemanggil mutasi mirip dengan *crossover*, hanya mengambil satu induk dan menghasilkan satu anak :

[c1] = mutation(p1,bounds,params)

Operator *crossover* harus mengambil keempat argumen, dua induk, batas ruang solusi, informasi tentang berapa banyak evolusi yang muncul dan pilihan khusus lain yang dibutuhkan. Mutasi hanya mengambil tiga argumen dan mengembalikan anak yang dihasilkan. Tabel 3 menampilkan fungsi-fungsi operator yang digunakan dalam penelitian ini, meliputi nama file yang digunakan, dan pilihan yang diambil oleh operator sebagai tambahan dari pilihan pertama. Modul listing program terdapat pada lampiran A.

Tabel 3 Fungsi-fungsi operator genetik khusus implementasi Matlab

Nama	File	Pilihan
Crossover Arithmatik	ArithXover.m	-
Crossover Heuristik	heuristicXover.m	bilangan retris (t)
Crossover sederhana	simleXover.m	-
Mutasi Boundary	boundary.m	-
Mutasi Uniform	UnifMut.m	-

Fungsi seleksi berfungsi menentukan individu yang bertahan dan berlanjut pada generasi berikutnya. Fungsi *ga* memanggil fungsi seleksi setiap generasi setelah semua anak yang baru dievaluasi untuk menciptakan populasi baru dari populasi lama. Oleh karena itu, semua rutin-rutin seleksi mengambil parameter populasi lama dari mana anggota-anggota dipilih dan pilihan tertentu untuk rutin seleksi khusus. Sesuai dengan materi kajian tentang maksimasi, dalam penelitian ini digunakan fungsi seleksi tipe roda rolet dengan nama file *roulette.m*. Langkah-langkah yang diperlukan untuk melakukan seleksi dengan metoda roda rolet adalah sebagai berikut (Davis, 1991: 14).

1) Jumlahkan *fitness* dari semua anggota populasi; hitung hasil total *fitness*,
2) Bangkitkan n suatu bilangan acak diantara 1 dan total *fitness*,
3) Laporkan anggota populasi yang memiliki *fitness*, tambahkan dengan *fitness* dari populasi lama, bila lebih besar atau sama dengan n.

Modul listing program terdapat pada lampiran A. Fungsi pemanggil dasar *ga* untuk seleksi adalah :

[newPop] = selectFunction(oldPop,options)

newPop adalah populasi baru yang terseleksi, *OldPop* adalah populasi baru, *options* adalah vektor untuk parameter pilihan yang lain.

Fungsi inisialisasi dan terminasi diperlukan pada bagian awal dan akhir dari suatu proses evolusi pemrograman genetik. Titik awal proses evolusi *ga* dimulai dengan pembangkitan *string* secara acak dalam ruang pencarian menggunakan fungsi inisialisasi. Ini merupakan prilaku yang ditetapkan oleh fungsi *ga*. Namun dimungkinkan untuk 'menyemai' (*seed*) individu-individu populasi inisial atau dengan membangkitkan penyelesaian-penyelesaian dalam bentuk yang lain. Algoritma genetik memungkinkan hal ini dengan pilihan parameter *startPop* yang menghasilkan *ga* dengan populasi awal eksplisit. Listing program terdapat pada lampiran A.

Fungsi terminasi menentukan kapan untuk menghentikan evolusi tersimulasi dan melapor populasi yang dihasilkan. Fungsi *ga* memanggil fungsi terminasi sekali setiap generasi setelah penerapan semua fungsi operator dan fungsi evaluasi untuk anak yang dihasilkan. Modul listing program terdapat pada lampiran A. Fungsi pemanggil berupa format :

Done = terminateFunction(options,bestPop,pop)

options merupakan verktor pilihan pertama terminasi yang menghasilkan generasi terbaru, *bestPop* adalah matriks individu terbaik dari masing-masing generasi yang ditemukan dan *pop* adalah populasi terbaru. Dalam penelitian ini digunakan fungsi terminasi jenis *maxgen* untuk mendapatkan nilai maksimum bertanda positif sesuai dengan tuntutan penyelesaian optimasi pemrograman linear.

Program utama untuk menjalankan pemrograman genetik dalam penelitian ini ditulis dalam satu file teks yang diberi nama 'Gaplin.m'. Program ini disusun berdasarkan uraian kajian prilaku fungsi yang terdapat pada modul pemrograman genetik dalam bentuk implementasi Matlab. Program utama digunakan untuk memberikan instruksi pada fungsi dasar *ga* agar memanggil fungsi-fungsi dan parameter yang dibutuhkan untuk penyelesaian optimasi pemrograman linear. Listing program Gaplin.m terdapat pada lampiran A. Dalam pengoperasiannya, Gaplin bekerja dengan sistem Float Genetic Algoritm (FGA).

Parameter pemrograman genetik berfungsi untuk mengendalikan proses evolusi *ga*. Parameter ini terdiri dari jumlah populasi dan generasi serta parameter operator genetik. *Parameter populasi dan generasi* mempengaruhi waktu kerja pemrograman genetik. Semakin banyak kromosom berada dalam satu populasi dari suatu generasi, semakin banyak waktu yang dibutuhkan untuk penghitungan fitness. Akibatnya beban komputasi akan menjadi besar. Jumlah populasi dan generasi

dengan ukuran yang besar dapat mengurangi nilai fitness dari kromosom terbaik. Untuk itu diperlukan variasi jumlah populasi dan generasi dalam melakukan uji program dengan metoda *trial and error* sehingga diperoleh suatu hasil dengan waktu dan beban komputasi yang relatif kecil.

Parameter operator genetik menentukan kinerja proses algoritma genetik. Parameter ini terdiri dari peluang *crossover* dan laju mutasi. Parameter *crossover* p_c menentukan peluang kromosom melakukan penyilangan. Bila nilai p_c yang diberikan terlalu besar, *string* yang merupakan kandidat solusi terbaik mungkin dapat hilang lebih cepat dari seleksi. Parameter laju mutasi (p_m) mendefinsikan seberapa acak perubahan terjadi pada populasi baru. Parameter laju mutasi mengendalikan operator mutasi dengan peluang lebih kecil dibandingkan peluang crossover. Untuk melakukan mutasi berganda pada kromosom digunakan sistem *Looped*. Sistem ini mengendalikan mutasi dalam *loop* dengan menggunakan nilai acak. *Loop* memutasikan kromosom sampai nilai acak lebih besar dari peluang mutasi yang diberikan. Semakin tinggi laju mutasi, semakin banyak mutasi melakukan perubahan.

Menu GA berfungsi untuk menyediakan dialog interaktif antara pemakai dengan program. Dalam menu GA terdapat daftar isian input GA yang harus dipenuhi oleh pemakai agar program dapat beroperasi. Listing program menu GA terdapat pada lampiran A. Dengan memanggil menu GA, program akan menampilkan daftar dialog isian sebagai berikut.

Daftar dialog isian input pemrograman genetik

MENU GA UNTUK MENEMUKAN NILAI OPTIMASI PADA MASALAH PEMROGRAMAN LINEAR	
INPUT PEMROGRAMAN ALGORITMA GENETIK UNTUK PEMROGRAMAN LINEAR (pc,pm,slc,jml gnr.,jml pop.,kfeval,bts,kvu)	
Masukkan Matriks Peluang *CrossOver* (pc)	=
Masukkan Matriks Laju Mutasi (pm)	=
Masukkan Koeffisien Peluang Seleksi (slc)	=
Masukkan Jumlah Generasi (jml gnr)	=
Masukkan Jumlah Populasi (jml pop)	=
Masukkan Matriks Koefisien Fungsi Evaluasi (kfeval)	=
Masukkan Matriks Batas Ruang Solusi (bts)	=
Masukkan Matriks Koeffisien Variabel Utama (kvu)	=
HASIL PROSES EVOLUSI ALGORITMA GENETIK :	

B. Alat Penelitian

Alat yang dibutuhkan untuk melakukan penelitian adalah satu set komputer dengan spesifikasi *processor pentium* II 300 MMX dan memori RAM 64 MHz.

C. Jalan Penelitian

Data hasil penelitian diperoleh dengan menjalankan penelitian sebagai berikut.

1. Perumusan model fungsi evaluasi dan ruang solusi sebagai input program genetik berdasarkan soal masalah pemrograman linear dengan langkah-langkah sebagai berikut :
   a Ubah pertidaksamaan pembatas menjadi persamaan bentuk standar dengan menambah variabel slack atau dengan mengurangkan pertidaksamaan pembatas dengan surplus variabel;

b Bentuk persamaan variabel pembatas tunggal dengan memodifikasi persamaan pembatas dengan menggunakan formula (2);

c Buat persamaan fungsi evaluasi yang dibentuk dari variabel slack/surplus;

d Tentukan nilai batas variabel fungsi evaluasi *melalui salah satu metoda identifikasi awal* berikut.

1) Bila pada persamaan pembatas bentuk standar terdapat variabel slack atau surplus variabel, tetapi bukan sebagai pembentuk persamaan fungsi evaluasi, maka variabel tersebut dianggap bernilai nol;

2) Bila kedua ruas persamaan fungsi objektif dan fungsi evaluasi yang diperoleh dari langkah c terdapat satu atau lebih variabel pembatas utama memiliki indeks yang sama, maka variabel slack dan surplus pada ruas fungsi evaluasi dianggap bernilai nol. Contoh $f(x)_{obj} = a_1.x_1 + a_2.x_2 + a_3.x_3 = f(x)_{eval} = b + b_1.x_2 + b_2.x_3 + b_3.x_4$, pada kedua ruas terdapat variabel pembatas utama x_2 dan x_3, maka variabel slack dan surplus x_4 pada ruas fungsi evaluasi nilainya dianggap nol;

3) Jumlahkan persamaan variabel pembatas tunggal yang diperoleh dari langkah b (tidak termasuk persamaan variabel pembatas tunggal slack/surplus). Bila hasilnya terdapat koefisien variabel bernilai nol, variabel tersebut dianggap bernilai nol.

Contoh $a_1x_1 + a_2x_2 = b + b_1x_3 + 0x_4$, variabel x_4 memiliki koefisien nol maka variabel tersebut dianggap bernilai nol;

4) Jika metoda identifikasi awal 1), 2) dan 3) tidak terpenuhi, maka substitusikan variabel slack/surplus dengan menyisakan satu buah variabel tidak bernilai nol bersama semua variabel pembatas dengan nilai sama dengan nol secara berurut-turut ke dalam persamaan pembatas variabel tunggal. Kemudian identifikasi nilai veriabel slack/surplus yang diperoleh terhadap tanda persamaan fungsi evaluasi untuk setiap variabel yang bersangkutan.

e Modifikasi persamaan fungsi objektif dan persamaan variabel pembatas tunggal sesuai dengan nilai yang diperoleh dari salah satu alternatif langkah d. Kemudian lakukan proses eliminasi dan substitusi persamaan variabel pembatas tunggal terhadap persamaan pembatas bentuk standar.

2. Menentukan parameter input program genetik berdasar representasi solusi *Float Genetic Algorithm* (FGA) meliputi; jumlah populasi dan generasi, peluang seleksi, peluang *crossover* dan peluang laju mutasi.
3. Memasukkan hasil langkah 1 dan 2 ke dalam listing program GaplinEval.m,
4. Menjalankan simulasi program Gaplin.m pada kondisi input normal untuk jumlah populasi 20 dan generasi 200. Proses evolusi program

akan melaporkan kondisi populasi awal, nilai fungsi evaluasi terbaik untuk setiap generasi, populasi akhir, nilai variabel ruang solusi yang menghasilkan fungsi evaluasi terbaik, beban komputasi dan grafik proses evolusi,

5. Melakukan variasi jumlah populasi dan generasi mulai titik terendah sampai ditemukan hasil yang sama seperti langkah 4,
6. Mensimulasi soal yang sama pada pemrograman metoda simpleks,
7. Mengnalisa nilai variabel pembatas, fungsi objektif dan beban komputasi yang diperoleh pada langkah 5 dan 6.

D. Analisa Data

Analisa data dalam penelitian ini menggunakan teknik komparasi antara hasil simulasi program genetik dan metoda simpleks. Hasil yang dikomparasikan terdiri atas nilai optimasi pemrograman linear dan beban komputasi yang dihasilkan. Bila diperoleh nilai optimasi dari kedua simulasi program sama, berarti pemrograman genetik dapat digunakan sebagai salah satu metoda penyelesaian optimasi pemrograman linear. Jika beban komputasi pemrograman genetik lebih kecil dibandingkan metode simpleks, berarti pemrograman genetik merupakan metoda yang efektif untuk penyelesaian pemrograman linear.

IV. HASIL PENELITIAN DAN PEMBAHASAN

Hasil penelitian ini berupa hasil eksekusi program genetik terpadu terhadap 7 buah soal uji. Input program terdiri dari fungsi evaluasi bersama ruang solusi dan parameter genetik yang dibutuhkan program meliputi jumlah populasi, jumlah generasi, koefisien seleksi, probabilitas *crossover* dan laju mutasi. Hasil pemodelan fungsi evaluasi bersama ruang solusi sesuai dengan uraian jalan penelitian point 1 untuk 7 soal uji disajikan pada tabel 4.

Tabel 4. Hasil pemodelan fungsi evaluasi dan ruang solusi

No.	Masalah Program Linear dan Transportasi Linear	Hasil Pemodelan Fungsi Evaluasi dan Ruang Solusi
1	2	3
1.	Maksimumkan : $f(x) = 4x_1 + 3x_2$ batasan : $2.x_1 + 3.x_2 \leq 6$ $-3.x_1 + 2.x_2 \leq 3$ $2.x_1 + x_2 \leq 4$ $0 \leq x_i \leq 2,\ i = 1, 2.$	$f(x)_{eval} = 10{,}5714 - 0{,}2855x_4$ ruang solusi $x_4 = [5{,}5\ \ 9]$
2.	Maksimumkan $f(x) = 5000x_1 + 4000x_2$ batasan : $10.x_1 + 15.x_2 \leq 150$ $20.x_1 + 10.x_2 \leq 160$ $30.x_1 + 10.x_2 \geq 135$ $x_1 - 3.x_2 \leq 0$ $x_1 + x_2 \geq 5$ x_1 dan $x_2 \geq 0$	$f(x)_{eval} = 23750 - 250x_6 + 4750x_7$ ruang solusi $x_6 = [16{,}5\ 70]$ $x_7 = [4{,}1429\ \ 6{,}5]$
3.	Maksimumkan $f(x) = x_1 - 2x_2 + 3x_3$ batasan : $x_1 + x_2 + x_3 \leq 70$ $x_1 - x_2 + x_3 \leq 20$ $3x_1 - x_2 - 2x_3 = -50$ $x_i \geq 0$	$f(x)_{eval} = 59 - 0{,}1x_4$ ruang solusi $x_4 = [0\ 43{,}3333]$
4.	Maksimumkan $f(x) = 2x_1 - 4x_2 + 4x_3$ batasan : $x_1 + 2x_2 + x_3 \leq 30$ $x_1 + x_2 + x_3 \geq 8$ $x_1 + x_2 = 10$ $x_i \geq 0$	$f(x)_{eval} = -120 + 10x_5$ ruang solusi $x_5 = [0\ 22]$

Sambungan tabel 4

1	2	3
5.	Minimumkan $f(x) = 100x_1 + 100x_2 + 100x_3$ batasan : $2x_1 + 2x_2 + x_3 \leq 22$ $2x_1 + x_2 + 2x_3 \geq 30$ $x_2 + 2x_3 \geq 25$ $x_i \geq 0$	$f(x) = 1650 - 33{,}33x_4$ ruang solusi $x_4 = [0\ \ 4{,}5]$
6.	Maksimumkan $f(x) = 5x_1 + 7x_2 + 10x_3$ batasan : $x_1 + x_2 + x_3 \geq 4$ $x_1 + 2x_2 + 4x_3 \geq 5$ $x_i \geq 0$	$f(x) = 22 - x_3$ ruang solusi $x_3 = [0{,}3333\ \ 1{,}25]$
7.	Maksimumkan $f(x) = x_1 + 2x_2 + 3x_3 + 4x_4$ batasan : $x_1 + 2x_2 + 2x_3 + 3x_4 \leq 20$ $2x_1 + x_2 + 3x_3 + 2x_4 \leq 20$ $x_i \geq 0$	$f(x) = 20 + x_3 + x_4$ ruang solusi $x_3 = [0\ \ 4]$ $x_4 = [0\ \ 4]$

Data pemodelan fungsi evaluasi bersama ruang solusi memperlihatkan fungsi evaluasi bersama ruang solusi pada umumnya dibentuk oleh variabel slack dan surplus variabel dengan jumlah yang lebih sedikit dibandingkan variabel utama. Upaya mereduksi jumlah variabel pembentuk fungsi evaluasi merupakan suatu usaha untuk mendapat beban komputasi relatif kecil saat program genetik dijalankan. Selain fungsi evaluasi bersama ruang solusi dibutuh input parameter operator genetik seperti disajikan pada tabel 5.

Tabel 5. Parameter input operator genetik

Parameter Operator Genetik	Koefisien Soal No. :						
	1	2	3	4	5	6	7
Jumlah Populasi	3	3	3	3	3	3	3
Jumlah Generasi	5	5	5	5	5	5	5
Seleksi	0,04	0,04	0,04	0,04	0,04	0,04	0,04
Crossover :							
Arithmatik	[1 0]	[1 0]	[1 0]	[1 0]	[1 0]	[1 0]	[1 0]
Heuristik	[1 3]	[1 3]	[1 3]	[1 3]	[1 3]	[1 3]	[1 3]
Simple Crossover	[1 0]	[1 0]	[1 0]	[1 0]	[1 0]	[1 0]	[1 0]
Mutasi :							
Mutasi *Boundary*	[2 0 0]	[2 0 0]	[2 0 0]	[2 0 0]	[2 0 0]	[2 0 0]	[2 0 0]
Mutasi Seragam	[2 0 0]	[2 0 0]	[2 0 0]	[2 0 0]	[2 0 0]	[2 0 0]	[2 0 0]

Parameter jumlah populasi dan generasi pada tabel 5 diperoleh secara *trial and error* saat melakukan uji program. Nilai ini mendukung evolusi program untuk menghasilkan nilai optimasi pada masalah pemrograman linear dengan beban komputasi yang relatif kecil. Dalam proses evolusi program secara normal sesuai dengan spesifikasi modul, dibutuh jumlah populasi dan generasi sebesar 20 dan 200. Dimungkinkannya program beroperasi diluar jumlah populasi dan generasi standar dengan menghasilkan suatu penyelesaian akhir, menunjukkan prilaku alamiah yang dimiliki algoritma genetik. Suatu sifat generasi yang sama akan muncul setelah beberapa generasi berikutnya. Selanjutnya parameter operator genetik meliputi koefisien seleksi, peluang *crossover* dan laju mutasi diguna nilai standar yang telah ditetapkan sesuai dengan spesifikasi modul. Variasi nilai parameter ini tidak terlalu banyak memberikan hasil terhadap pencapaian tujuan penelitian. Namun sangat dibutuhkan dalam menjalankan program.

Berdasarkan kedua jenis data input tersebut, saat program dijalankan diperoleh hasil eksekusi pemrograman genetik. Dalam hasil eksekusi dilaporkan informasi tentang populasi awal (*startpop*), populasi akhir (*endpop*), nilai fungsi evaluasi terbesar pada setiap generasi, nilai variabel pembatas untuk penyelesaian fungsi evaluasi dan *info trace* yang memperlihatkan nilai maksimum dan rata-rata fungsi evaluasi serta kondisi evolusi program saat mencapai konvergen. Data lengkap hasil eksekusi program dilampirkan pada lampiran B.

Selanjutnya diuraikan pembahasan mengenai proses evolusi program. Dari 7 buah soal uji, dipilih satu buah soal uji sebagai materi pembahasan, yaitu soal nomor 2. Untuk soal uji yang lain tidak dibahas karena bentuk fungsi evaluasi bersama ruang solusi pada prinsipnya memiliki pola yang sama dan diproses dengan program yang sama. Data populasi awal hasil eksekusi program genetik untuk soal uji 2 adalah sebagai berikut :

Tabel 6. Populasi awal soal uji 2

Populasi	Nilai *fitness* variabel :		Nilai *Fitness* Fungsi Evaluasi
	x_6	x_7	
1	44,517	5,9903	41075
2	50,153	6,1217	40290
3	42,123	4,6343	35230

Populasi awal pada tabel 6 dibangkitkan secara acak melalui fungsi inisialisasi berdasarkan input ruang solusi dan jumlah populasi. Masing-masing kromosom akan diseleksi melalui seleksi jenis roda rolet untuk menentukan peluang kesempatan terpilih sebagai induk pada generasi berikutnya. Kesempatan ini ditentukan oleh persentase terbesar dari masing-masing nilai *fitness* fungsi evaluasi terhadap total *fitness*, seperti disajikan pada tabel 7.

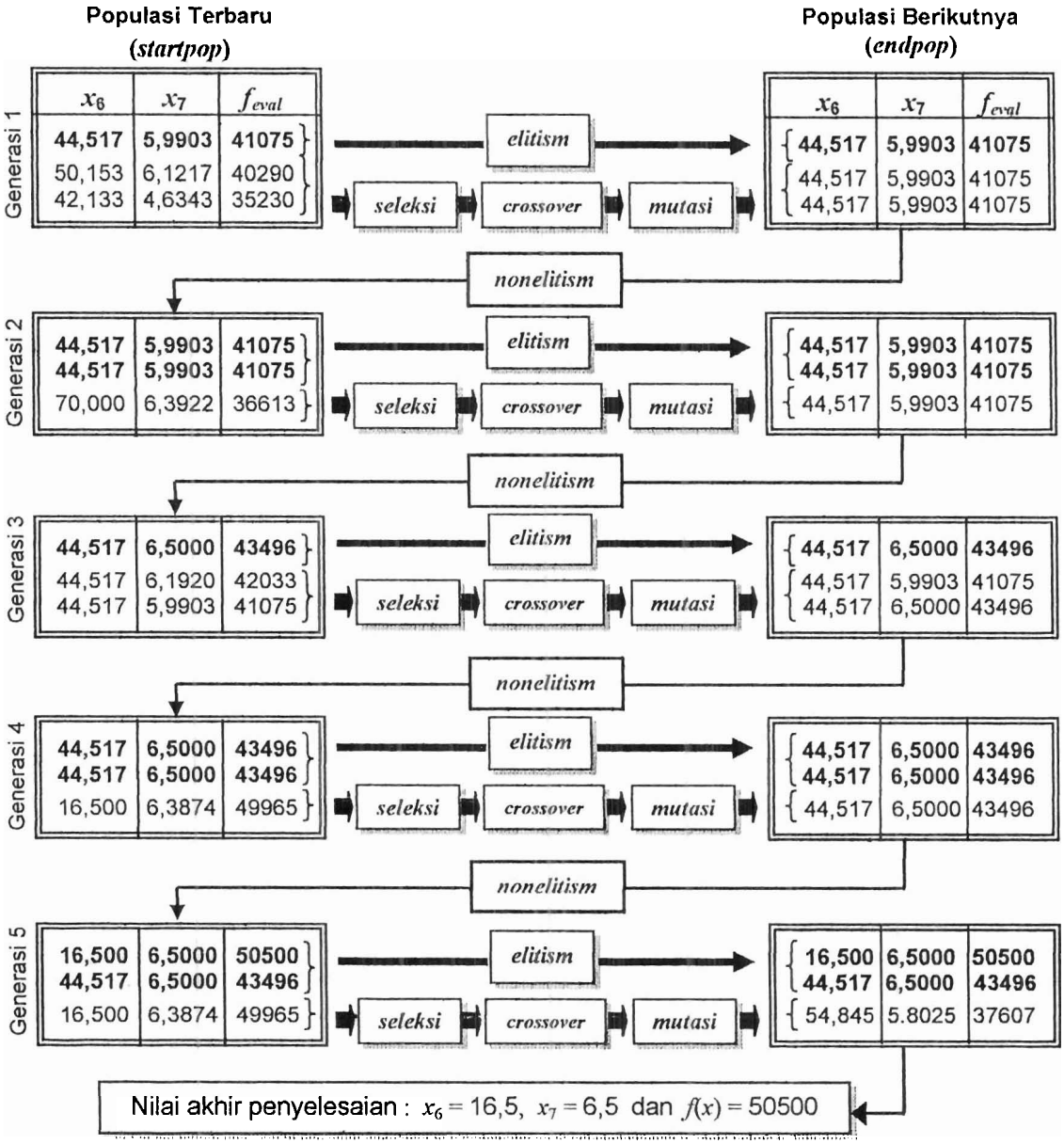
Tabel 7. Peluang terpilih sebagai induk

Populasi	Nilai *fitness*		Kesempatan Terpilih sebagai induk
	x_6	x_7	
1.	44,517	5,9903	35,23%
2	50,153	6,1217	34,56%
3	42,133	4,6343	30,21%

Pada tabel 7 terlihat kromosom pada urut 1 lajur populasi memiliki peluang terbesar terpilih sebagai induk. Kromosom dengan urut 2 memiliki kesempatan 34,56% terpilih sebagai induk untuk menghasilkan populasi baru dan begitu seterusnya untuk kromosom-kromosom yang lain.

Untuk mencegah individu-individu terbaik tidak hilang pada tingkat berikutnya, nilai *fitness* harus dihitung ulang pada setiap perpindahan tingkat. Mengingat fungsi *fitness* yang berbeda antar tahapan, melalui metoda *elitism* dipelihara sejumlah individu terbaik pada setiap populasi baru yang dihasilkan. Kemudian setiap anggota populasi terbaik disalin sebagai anggota individu untuk populasi berikutnya. Metoda ini juga menerapkan dua individu terbaik untuk disilangkan dengan generasi yang ada. Proses metoda *elitism* secara rinci untuk soal uji 2 diperlihatkan pada gambar 9 halaman 53. Angka-angka yang dicetak tebal dalam kotak populasi terbaru adalah di antara individu terbaik yang dipelihara dan mendapat kesempatan menjadi induk.

Data populasi terbaru dan populasi berikutnya dari setiap generasi pada gambar 9 diperoleh setelah program dijalankan secara bertahap. Pada generasi pertama, terdapat satu individu terbaik yang memenuhi metoda *elitism* dan dipilih menjadi induk untuk disilangkan dengan individu yang lain. Individu ini dipertahankan dan disalin sebagai anggota populasi berikutnya. Setelah penghitungan ulang, terpilih dua individu terbaik memiliki nilai *fitness* yang sama yaitu 41075 sebagai populasi terbaru untuk generasi kedua yang dipelihara dan disalin sebagai anggota populasi berikutnya.



Gambar 9 . Proses evolusi program genetik

Satu individu terbaik dari generasi pertama (41075) yang termasuk dalam individu terbaik dari generasi kedua terpilih sebagai induk untuk disilangkan dengan anggota populasi yang lainnya. Terhadap anak yang baru

dilahirkan bersama individu yang dipelihara, diadakan penghitungan ulang untuk menentukan anggota populasi terbaru generasi ke tiga. Proses ini menghasilkan satu individu terbaik, yaitu 43496 menjadi anggota populasi generasi ketiga yang akan dipelihara dan disalin sebagai anggota populasi berikutnya. Individu dengan nilai *fitness* 41075 seharusnya memiliki kesempatan 3 kali terpilih sebagai induk. Setelah diadakan penghitungan ulang, populasi tersebut mendapat kesempatan menjadi induk hanya satu kali dan dipertahankan sebagai populasi terbaru generasi ketiga. Hal ini mungkin disebabkan tidak dapat menghasilkan keturunan setelah satu kali menjadi induk sebagaimana dijelaskan dalam contoh pemilihan kromoson induk oleh Robert yang telah dikemukakan dalam BAB II halaman 15. Setelah diadakan penghitungan ulang terhadap anak yang baru dilahirkan bersama individu terbaik yang dipelihara, terdapat dua individu terbaik dengan nilai *fitness* yang sama, yaitu 43496 menjadi anggota generasi keempat.

Pada generasi keempat terdapat dua individu terbaik yang dipelihara dan disalin sebagai anggota populasi berikutnya. Satu di antara individu terbaik dengan nilai *fitness* 43496 berasal dari anggota populasi generasi sebelumnya. Kedua individu ini disilangkan dengan anggota generasi yang ada. Anak-anak yang baru dilahir bersama individu terbaik dilakukan penghitungan ulang untuk menentukan anggota populasi generasi kelima. Proses ini menghasilkan individu dengan nilai *fitness* 50500 dan 43496

menjadi anggota populasi terbaru generasi kelima. Setelah melalui proses tahapan evolusi meliputi seleksi, *crossover* dan mutasi terhadap populasi terbaru generasi kelima, terpilih individu dengan nilai *fitness* 50500 sebagai penyelesaian akhir soal uji 2.

Dengan terpilihnya suatu populasi sebagai penyelesaian akhir, maka proses evolusi program berhenti. Hal ini ditunjukkan oleh data standart deviasi bernilai nol sebagai indikasi evolusi program telah mencapai konvergen, seperti diperlihatkan pada tabel 8.

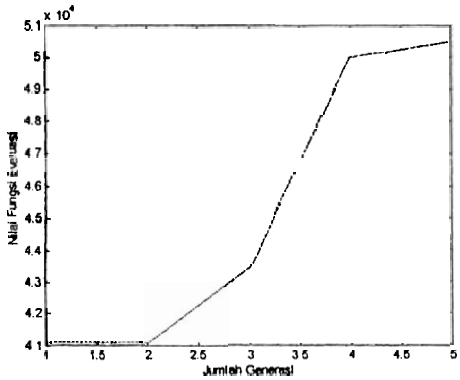
Tabel 8. Data informasi *trace* soal uji 2

Generasi ke :	Nilai maks $f(x)_{eval}$	Nilai rata-rata $f(x)_{eval}$	Std
1.	41075	38865	3172
2	41075	39587	3576
3	43496	42201	1219
4	49965	45652	3735
5	50500	47987	0

Data informasi *trace* tabel 8 selain menampilkan nilai standar deviasi setiap generasi, juga diinformasi nilai maksimum dan rata-rata fungsi evaluasi untuk setiap proses evolusi pada setiap generasi. Nilai maksimum fungsi evaluasi adalah nilai *fitness* terbaik dari setiap generasi yang disortir dari setiap populasi terbaru. Nilai rata-rata fungsi evaluasi adalah nilai yang diperoleh dari hasil penjumlahan seluruh nilai fungsi *fitness* pada setiap generasi dibagi dengan jumlah populasi yang berhubungan.

Dalam data nilai maksimum fungsi evaluasi dari generasi pertama menuju generasi kedua tidak terjadi perbaikan nilai *fitness*. Dari generasi

ketiga sampai kelima terjadi perbaikan nilai fungsi *evaluasi.* Berdasarkan data ini, program melaporkan hasil proses evolusi dalam bentuk grafik seperti diperlihatkan pada gambar 10.



Gambar 10. Grafik proses evolusi genetik

Berikut ditampilkan ringkasan hasil eksekusi program untuk 7 buah soal uji, seperti disajikan pada tabel 9.

Tabel 9. Ringkasan hasil eksekusi pemrograman genetik

N0.	Fungsi Evaluasi	Ruang Solusi	Hasil Eksekusi	
			x	$f(x)_{eval}$
1	2	3	4	5
1.	$f(x)_{eval} = 10{,}5714 - 0{,}2855x_4$	$x_4 = [5{,}5\ \ 9]$	x_1=1,5003, x_2=1,0001	9,0012
2.	$f(x)_{eval} = 23750 - 250x_6 + 4750x_7$	x_6=[16,50 70] x_7=[4,1429 6,5]	$x_1 = 4{,}5$ $x_2 = 7{,}0$	50500
3.	$f(x)_{eval} = 59 - 0{,}1x_4$	x_4 =[0 43,3333]	$x_1 = 0$, x_2 =3,3333 x_3 =23,333	63,333
4.	$f(x)_{eval} = -120 + 10x_5$	x_5 =[0 22]	$x_1 = 10$, $x_2 = 0$, x_3 =20	100
5.	$f(x)_{eval} = 1650 - 33{,}33x_4$	x_4 =[0 4,5]	$x_1 = 2{,}5$, $x_2 = 0$ x_3 =12,4999	1500
6.	$f(x)_{eval} = 22 - x_3$	x_3=[0,3333 1,25]	$x_1 = 3{,}6667$, $x_2 = 0$ $x_3 = 0{,}3333$	21,667
7.	$f(x)_{eval} = 20 + x_3 + x_4$	$x_3 = [0\ \ 4]$ $x_4 = [0\ \ 4]$	$x_1 = 0$, $x_2 = 0$, $x_3 = 4$ $x_4 = 4$	28

Dalam lajur ruang solusi dan lajur hasil eksekusi tabel 9 terdapat perbedaan indeks variabel. Nilai variabel hasil eksekusi dengan indeks yang sama dengan indeks variabel pada lajur ruang solusi merupakan hasil evolusi program. Nilai variabel dengan indeks yang berbeda merupakan hasil penghitungan komputasi biasa berdasarkan nilai variabel yang diperoleh dari hasil evolusi program. Koefisien variabel ini dengan inisial *kko* seperti terdapat dalam file teks listing program GaplinEval.m pada lampiran A, diperoleh dari proses perumusan pemodelan fungsi evaluasi dan ruang solusi yang telah dikemukakan pada uraian jalan penelitian point 1.

Data tabel 9 memperlihatkan bahwa pemrograman genetik sebagai alat untuk menyelesaikan masalah pemrograman linear telah menghasilkan penyelesaian sesuai yang diharapkan. Data hasil eksekusi pemrograman genetik juga telah diuji dengan metoda simpleks dengan menggunakan listing program fungsi barnes (Lindfiel, 1995: 274). Listing program dan hasilnya terdapat pada lampiran C. Perbandingan hasil penyelesaian dan beban komputasi antara pemrograman genetik dengan metoda simpleks untuk 7 soal uji ditampilkan pada tabel 10 halaman 58. Hasil eksekusi kedua program memperlihatkan fungsi objektif ($f(x)$) memiliki nilai yang sama. Perbedaan nilai yang relatif kecil pada salah satu variabel pembatas mungkin disebabkan oleh pembulatan angka dibelakang koma yang dilakukan program. Dilihat dari beban komputasi, pemrograman genetik lebih kecil dibandingkan metoda simpleks. Dalam bentuk nilai rata-rata untuk 7 soal uji yang dilakukan, pemrograman genetik 9 kali lebih baik dibandingkan metoda simpleks.

Tabel 10. Perbandingan hasil eksekusi antara pemrograman genetik dengan metoda simpleks untuk 7 buah soal uji

Soal No.	Hasil Yang Diperoleh :					
	Pemrograman Genetik			Metoda Simpleks		
	x	$f(x)$	Bbn. Komp.	x	$f(x)$	Bbn. Komp.
1	2	3	4	5	6	7
1.	$x_1 = 1{,}5003$ $x_2 = 1{,}0001$	9,0012	639	$x_1 = 1{,}5000$ $x_2 = 1{,}0000$	9,0000	5262
2.	$x_1 = 4{,}5$ $x_2 = 7{,}5$	50500	727	$x_1 = 4{,}5$ $x_2 = 7{,}5$	50500	9771
3	$x_1 = 0$, $x_2 = 3{,}3333$ $x_3 = 23{,}333$	63,333	638	$x_1 = 0$, $x_2 = 3{,}3333$ $x_3 = 23{,}333$	63,333	5281
4	$x_1 = 10$, $x_2 = 0$, $x_3 = 20$	100	628	$x_1 = 10$, $x_2 = 0$, $x_3 = 20$	100	4387
5.	$x_1 = 2{,}5$ $x_2 = 0$ $x_3 = 12{,}499$	1500	639	$x_1 = 2{,}5$ $x_2 = 0$ $x_3 = 12{,}5$	1500	7468
6.	$x_1 = 3{,}667$ $x_2 = 0$ $x_3 = 0{,}333$	21,667	620	$x_1 = 3{,}667$ $x_2 = 0$ $x_3 = 0{,}333$	21,667	4938
7.	$x_1 = 0$ $x_2 = 0$ $x_3 = 4$ $x_4 = 4$	28	698	$x_1 = 0$ $x_2 = 0$ $x_3 = 4$ $x_4 = 4$	28	6278

Berdasarkan data yang diperoleh, dapat diinterpretasikan sebagai hasil penelitian bahwa metoda algoritma genetik dapat digunakan untuk menemukan nilai optimasi pada masalah pemrograman linear. Secara simulasi program, algoritma genetik memiliki unjuk kerja yang lebih baik dibandingkan metoda simpleks dalam kaitanya terhadap beban komputasi, *bila fungsi objektif bersama pembatas kombinasi linear pada masalah pemrograman linear telah diubah menjadi bentuk fungsi evaluasi dan ruang*

solusi. Dengan demikian pertanyaan perumusan masalah yang dimajukan dan tujuan penelitian yang telah ditetapkan sebagaimana diuraikan pada BAB I, dapat terwujud.

V. KESIMPULAN DAN SARAN

A. Kesimpulan

Berdasarkan hasil penelitian dan pembahasan dapat disimpulkan bahwa algoritma genetik dapat digunakan sebagai salah satu metoda untuk menemukan nilai optimasi pada masalah pemrograman linear, *bila fungsi objektif bersama pembatas pada permasalahan pemrograman linear telah diubah menjadi fungsi evaluasi dan ruang solusi.* Selanjutnya terdapat tiga aspek penting untuk menemukan nilai optimasi dalam proses evolusi algoritma genetik yang menghasilkan beban komputasi relatif kecil dibandingkan metoda simpleks. Ketiga aspek tersebut terdiri atas pemodelan fungsi evaluasi bersama ruang solusi dan representasi solusi yang digunakan serta jumlah populasi dan generasi.

Fungsi evaluasi bersama ruang solusi sebagai fungsi *fitness* merupakan kebutuhan pokok untuk menjalankan pemrograman genetik. Fungsi ini menjabarkan kemampuan program dalam menyelesaikan masalah sesuai kreteria yang ditetapkan. Setiap masalah yang berbeda, fungsi evaluasi dan ruang solusi harus dibuat model perumusan. Pemodelan fungsi evaluasi dan ruang solusi untuk masalah pemrograman linear yang memenuhi sifat nonlinear $D = \prod_{k=1}^{q} \langle \ell_k, r_m \rangle$ dilakukan dengan menggunakan formula $X^1 = A_1^{-1} \cdot b - A_1^{-1} \cdot A_2 \cdot X^2$. Variabel X^1 adalah variabel pembatas ruang solusi yang terdiri atas $X^1 = (x_{i_1}, \cdots, x_{i_p})$ dan

nilainya ditentukan oleh X^2. Variabel X^2 umumnya terdiri dari variabel slack atau surplus variabel yang jumlahnya lebih sedikit dibanding variabel X^1. Fungsi evaluasi dengan persamaan $f(x)_{eval} = konstanta + f(x_1, x_2, \ldots, x_k)$ diperoleh dengan mensubstitusi variabel ruang solusi X^1 ke persamaan fungsi objektif. Selanjutnya melalui eliminasi dan substitusi terhadap sejumlah persamaan variabel pembatas utama X^1, didapatkan himpunan titik cembung ruang solusi $\{x_1 = [\ell_1 \quad r_1], \cdots, x_k = [\ell_k \quad r_k]\}$ sebagai ruang pelacakan bagi algoritma genetik dalam menemukan solusi optimal. Metoda ini telah berhasil menemukan variabel ruang solusi yang bersifat nonlinear sebagai pembentuk fungsi evaluasi dengan jumlah yang minimal. Semakin minimal jumlah variabel yang disimulasikan pada proses evolusi pemrograman genetik, semakin kecil beban komputasi yang dihasilkan.

Pemilihan representasi solusi yang sesuai dengan sifat dan kinerja jenis operator genetik sangat menentukan dalam penyelesaian masalah. Representasi solusi diperlukan oleh algoritma genetik untuk mendapatkan gambaran tiap individu sebagai titik fokus perhatian, sehingga dapat ditentukan operator yang akan digunakan. Untuk masalah optimasi fungsi dengan melibatkan ruang solusi dalam bentuk bilangan nyata, pemanfaatan representasi solusi sistem *Float Genetic Algorithm* (FGA) merupakan pilihan yang tepat. Sistem ini tidak membutuhkan gambaran yang lengkap dari masalah yang dipecahkan, hanya dibutuhkan satu

fungsi evaluasi bersama ruang solusi. Memiliki kemampuan untuk menemukan hasil penyelesaian dengan ketepatan yang lebih tinggi dan tidak mudah terjebak dalam minima lokal pada masalah yang kompleks. Beban komputasi dapat diturunkan karena selama proses evolusi tidak terjadi konversi bilangan biner ke desimal.

Jumlah populasi dan generasi memiliki hubungan yang berbanding lurus terhadap beban komputasi. Semakin besar jumlah generasi dan populasi, semakin besar beban komputasi yang dihasilkan. Berdasarkan prilaku alamiah yang dimiliki algoritma genetik bahwa sifat individu dalam suatu populasi dari suatu generasi akan muncul setelah beberapa generasi berikutnya. Dengan prilaku ini, penyelesaian yang dihasilkan oleh algoritma genetik akan muncul di antara beberapa titik populasi dan generasi. Untuk itu, diperlukan variasi jumlah populasi dan generasi secara *trial and error* guna menemukan nilai yang relatif kecil, sehingga penyelesaian yang ditemukan menghasilkan beban komputasi yang relatif kecil.

B. Saran

Algoritma genetik memiliki banyak kelebihan yang dapat diandalkan sebagai alat untuk memecahkan masalah yang kompleks tanpa membutuhkan algoritma tertuntun secara matematik. Untuk itu telah sewajarnya algoritma genetik dikembangkan keberadaanya dimulai dari

institusi pendidikan. Melalui penelitian yang telah dilakukan, diharapkan mampu membangkitkan minat untuk melakukan penelitian pada bidang yang lain dalam rangka mendalami dan mengembangkan algoritma genetik lebih lanjut.

Kemudian bertitik tolak dari pelaksanaan penelitian, belum ditemukan metoda pemodelan fungsi evaluasi dan ruang solusi yang dipandang efektif. Hasil pemodelan ini masih bersifat parsial untuk kasus perkasus. Oleh karena itu, dibutuhkan penelitian lebih lanjut guna menemukan satu metoda pemodelan fungsi evaluasi dan ruang solusi untuk semua kasus yang berbeda dalam penyelesaian pemrograman linear.

DAFTAR PUSTAKA

Davis, Lawrence (edit). 1991. Handbook of Genetic Algorithm, New York: Van Nostrand Reinhold.

Goldberg, Davit E. 1989. Genetic Algorithms in Search, Optimization, and Machine Learning, Reading. MA: Addison-Wesley.

Hassoun, Mohamad H. 1995. Fundamentals of Artificial Neural Networks, London: A Bradford Book The MIT Press Cambridge, Massachusetts.

Houck, dkk. tt. A Genetic Algorithm for Function Optimization: A Matlab Implementation, North Carolina State University: Internet chouck, jjoine, kay @eos.ncsu.edu.

Jang, dkk. 1997. Neuro-Fuzzy and Soft Computing: A Computational Approach to Learning and Machine Intelligensi, USA: Prentice Hall International Inc.

Lindfiel, G., dan Penny, J. 1995. Numerical Methods Using Matlab. New York: Eliss Horwood Limited.

Man, dkk. 1996. "Genetic Algorithms: Consepts and applications", IEEE Trans. Ind. electron., vol. 43, no. 5, Okt.,pp. 519-531.

Michalewicz, Zbigniew. 1995. Genetic Algorithms + Data Structures = Evolution Programs, USA: Springer.

Robert Ladd, Scot. 1996. Genetic Algorithm in C++, New York: M&T Books.

Lampiran A. Listing program algoritma genetik implementasi Matlab

1. Gaplin.m

%Digunakan untuk memberikan input program fungsi dasar GA sesuai dengan %masalah optimasi pemrograman linear, yang terdiri dari operator seleksi roda %rolet, crossover arithXover, heuristicXover, simpleXover, mutasi boundary, %mutasi uniform, jumlah generasi dan populasi serta terminasi maxgen.
%Dalam listing program ini juga memuat laporan hasil eksekusi program, yang %terdiri dari populasi awal dan akhir sebagai kandidat solusi, solusi terbesar %untuk setiap generasi, hasil penghitungan nilai variabel pembatas fungsi %evaluasi dan beban komputasi serta data grafik solusi terbesar pada setiap %generasi.

```
global inpxOpts inpmOpts inptermOps inpselectOps inpbounds inpjumpop
kko;
GaplinEval(0);
global bounds
% Setting the seed to the same for binary
rand('seed',156789)
flops(0);
%Operator Crossover
xFns = 'arithXover heuristicXover simpleXover';
xOpts = inpxOpts;
% Operator Mutasi
mFns = 'boundaryMutation unifMutation';
mOpts = inpmOpts;
% Operator Terminasi
termFns = 'maxGenTerm';
termOps = inptermOps; % Jumlah Generasi
% Seleksi Fungsi
selectFn = 'roulette'
selectOps = inpselectOps;
%Fungsi Evaluasi
evalFn = 'GaplinEval';
evalOps = [];
%type GaplinEval
% Batas ruang fungsi evaluasi
bounds = inpbounds;
% GA Options [epsilon float/binar display]
gaOpts=[1e-6 1 1];
% Generate an intialize population of size jumpop
jumpop=inpjumpop;
startPop = initializega(jumpop,bounds,'GaplinEval',[1e-6 1])
 [x endPop bestPop trace]=ga(bounds,evalFn,evalOps,startPop,gaOpts,..
    termFns,termOps,selectFn,selectOps,xFns,xOpts,mFns,mOpts);
beban=flops;
endPop
```

bersambung...

```
bestPop
% x adalah solusi terbaik yang ditemukan
x;
nn=max(size(x));
for i=1:nn-1
   fprintf('\nNilai Variabel sol(%1.0f) =%8.4f\',i,x(i));
end
ko=kko;
x=[1 x];
nn=max(size(x));
xxx=ko*x(1:nn-1)';
fprintf('\nNilai Fungsi Evaluasi =%8.4f',x(nn));
nn=max(size(xxx));
for i=1:nn
   fprintf( '\nNilai Variabel x%d =%8.4f',i,xxx(i) );
end
fprintf( '\nBeban Komputasi = %d\n',beban);
trace
pause
% Plot the best over time
clf
plot(trace(:,1),trace(:,2));
```

3. *Crossover*

a. arithXover.m

```
function [c1,c2] = arithXover(p1,p2,bounds,Ops)
% Pick a random mix amount
a = rand;
% Create the children
c1 = p1*a     + p2*(1-a);
c2 = p1*(1-a) + p2*a;
```

b. heuristicXover.m

```
function [c1,c2] = heuristicXover(p1,p2,bounds,Ops)
retry=Ops(3);                          % Number of retries
i=0;
good=0;
b1=bounds(:,1)';
b2=bounds(:,2)';
numVar = size(p1,2)-1;
% Determine the best and worst parent
if(p1(numVar+1) > p2(numVar+1))
  bt = p1;
  wt = p2;
else
  bt = p2;
  wt = p1;
end
```

bersambung...

```
while i<retry
% Pick a random mix amount
  a = rand;
% Create the child
  c1 = a * (bt - wt) + bt;
% Check to see if child is within bounds
  if (c1(1:numVar) <= b2 & (c1(1:numVar) >= b1))
    i = retry;
    good=1;
  else
    i = i + 1;
  end
end
 % If new child is not feasible just return the new children
if(~good)
  c1 = wt;
end
% Crossover functions return two children therefore return the best
% and the new child created
c2 = bt;
```

c. simpleXover.m

```
function [c1,c2] = simpleXover(p1,p2,bounds,Ops)
numVar = size(p1,2)-1;                % Get the number of variables
% Pick a cut point randomly from 1-number of vars
cPoint = round(rand * (numVar-2)) + 1;
c1 = [p1(1:cPoint) p2(cPoint+1:numVar+1)]; % Create the children
c2 = [p2(1:cPoint) p1(cPoint+1:numVar+1)];
```

3. Mutasi

a. boundaryMutation.m

```
function [parent] = boundaryMutate(parent,bounds,Ops)
numVar = size(parent,2)-1;            % Get the number of variables
% Pick a variable to mutate randomly from 1-number of vars
mPoint = round(rand * (numVar-1)) + 1;
b = round(rand)+1;                 % Pick which bound to move to
newValue = bounds(mPoint,b);    % Now mutate that point
parent(mPoint) = newValue;            % Make the child
```

b. unifMutation.m

```
function [parent] = uniformMutate(parent,bounds,Ops)
df = bounds(:,2) - bounds(:,1);       % Range of the variables
numVar = size(parent,2)-1;            % Get the number of variables
% Pick a variable to mutate randomly from 1-number of vars
mPoint = round(rand * (numVar-1)) + 1;
newValue = bounds(mPoint,1)+rand * df(mPoint); % Now mutate that
point
parent(mPoint) = newValue;            % Make the child
```

4. Terminasi (maxGenTerm.m)

```
function [done] = maxGenTerm(ops,bPop,endPop)
currentGen = ops(1);
maxGen     = ops(2);
done       = currentGen >= maxGen;
```

5. Seleksi Fungsi (roulette.m)

```
function[newPop] = roulette(oldPop,options)
%Get the parameters of the population
numVars = size(oldPop,2);
numSols = size(oldPop,1);
%Generate the relative probabilites of selection
totalFit = sum(oldPop(:,numVars));
prob=oldPop(:,numVars) / totalFit;
prob=cumsum(prob);
rNums=sort(rand(numSols,1));            %Generate random numbers
%Select individuals from the oldPop to the new
fitIn=1;newIn=1;
while newIn<=numSols
  if(rNums(newIn)<prob(fitIn))
    newPop(newIn,:) = oldPop(fitIn,:);
    newIn = newIn+1;
  else
    fitIn = fitIn + 1;
  end
end
```

6. Fungsi Evaluasi (GaplinEval.m)

```
function [sol,val] = GaplinEval(sol,options);
 global D6
 ND6=max(size(D6));
 val=D6(1);
   for i=2:ND6
 val =val + D6(i)*sol(i-1);
 end
```

7. Menu GA.m

```
clc;
disp(' ');
disp('MENU UTAMA GA UNTUK MENEMUKAN NILAI OPTIMASI PADA MASALAH PEMROGRAMAN LINEAR');
disp(' ');
disp('
==========================================================================');
disp('   INPUT PEMROGRAMAN ALGORITMA GENETIK UNTUK PEMROGRAMAN LINEAR       ');
disp('       (pc,pm,slc,jml gnr.,jml pop.,kfeval,bts,kvu)                   ');
disp('
==========================================================================');
```

bersambung...

```
inpxOpts=input('Masukkan Matriks Peluang CrossOver(pc)        =       ');
inmOpts=input('Masukkan Matriks Laju Mutasi(pm)               =       ');
inpselectOps =input('Masukkan Koeffisien Peluang Seleksi(slc) =       ');
inpjumpop =input('Masukkan Jumlah Generasi(jml gnr)           =       ');
inptermOps =input('Masukkan Jumlah Populasi (jml pop)         =       ');
D6=input('Masukkan Matriks Koeffisien Fungsi Evaluasi(kfeval) =       ');
inpbounds=input('Masukkan Matriks Batas Ruang Solusi (bts)    =       ');
kko=input('Masukkan Matriks Koeffisien Variabel Utama (kvu)   =       ');
disp('=================================================================');
disp(' ');
disp('     HASIL PROSES EVOLUSI ALGORITMA GENETIK');
pause
gaplin
```

8. Fungsi dasar GA (ga.m)

```
function [x,endPop,bPop,traceInfo] = ga(bounds,evalFN,evalOps,....
startPop,opts, termFN,termOps,selectFN,selectOps,xOverFNs,....
xOverOps,mutFNs,mutOps)
n=nargin;
if n<2 | n==6 | n==10 | n==12
  disp('Insufficient arguements')
end
if n<3 %Default evalation opts.
  evalOps=[];
end
if n<5
  opts = [1e-6 1 0];
end
if isempty(opts)
  opts = [1e-6 1 0];
end
if any(evalFN<48) %Not using a .m file
  if opts(2)==1 %Float ga
    e1str=['x=c1; c1(xZomeLength)=', evalFN ';'];
    e2str=['x=c2; c2(xZomeLength)=', evalFN ';'];
  else %Binary ga
    e1str=['x=b2f(endPop(j,:),bounds,bits); endPop(j,xZomeLength)=',...
    evalFN ';'];
  end
else %Are using a .m file
  if opts(2)==1 %Float ga
    e1str=['[c1 c1(xZomeLength)]=' evalFN '(c1,[gen evalOps]);'];
    e2str=['[c2 c2(xZomeLength)]=' evalFN '(c2,[gen evalOps]);'];
else %Binary ga
    e1str=['x=b2f(endPop(j,:),bounds,bits);[x v]=' evalFN ...
    '(x,[gen evalOps]); endPop(j,:)=[f2b(x,bounds,bits) v];'];
  end
end
  if n<6 %Default termination information
    termOps=[100];
```

bersamabung...

```
    termFN='maxGenTerm';
end
if n<12 %Default muatation information
  if opts(2)==1 %Float GA
  mutFNs=['boundaryMutation multiNonUnifMutation nonUnifMutation....
          unifMutation'];
    mutOps=[4 0 0;6 termOps(1) 3;4 termOps(1) 3;4 0 0];
  else %Binary GA
    mutFNs=['binaryMutation'];
    mutOps=[0.05];
  end
end
if n<10 %Default crossover information
  if opts(2)==1 %Float GA
    xOverFNs=['simpleXover arithXover heuristicXover '];
    xOverOps=[2 0;2 3;2 0];
  else %Binary GA
    xOverFNs=['simpleXover'];
    xOverOps=[0.6];
  end
end
if n<9 %Default select opts only i.e. roullete wheel.
  selectOps=[];
end
if n<8 %Default select info
  selectFN=['normGeomSelect'];
  selectOps=[0.08];
end
if n<6 %Default termination information
  termOps=[200];
  termFN='maxGenTerm';
end
if n<4 %No starting population passed given
  startPop=[];
end
if isempty(startPop) %Generate a population at random
  %startPop=zeros(80,size(bounds,1)+1);
  startPop=initializega(80,bounds,evalFN,evalOps,opts(1:2));
end
if opts(2)==0 %binary
  bits=calcbits(bounds,opts(1));
end
xOverFNs=parse(xOverFNs);
mutFNs=parse(mutFNs);
xZomeLength = size(startPop,2);   %Length of the xzome=numVars+fittness
numVar     = xZomeLength-1;       %Number of variables
popSize    = size(startPop,1);    %Number of individuals in the pop
endPop     = zeros(popSize,xZomeLength); %A secondary population matrix
c1         = zeros(1,xZomeLength);       %An individual
c2         = zeros(1,xZomeLength);       %An individual
numXOvers = size(xOverFNs,1);    %Number of Crossover operators
```

bersambung...

```
numMuts    = size(mutFNs,1);              %Number of Mutation operators
epsilon    = opts(1);            %Threshold for two fittness to differ
oval       = max(startPop(:,xZomeLength)); %Best value in start pop
bFoundIn   = 1;                  %Number of times best has changed
done       = 0;                  %Done with simulated evolution
gen        = 1;                  %Current Generation Number
collectTrace = (nargout>3);      %Should we collect info every gen
floatGA    = opts(2)==1;         %Probabilistic application of ops
display    = opts(3);            %Display progress
while(~done)
  %Elitist Model
  [bval,bindx] = max(startPop(:,xZomeLength)); %Best of current pop
  best =  startPop(bindx,:);
  if collectTrace
    traceInfo(gen,1)=gen;                    %current generation
    traceInfo(gen,2)=startPop(bindx,xZomeLength);       %Best fittness
    traceInfo(gen,3)=mean(startPop(:,xZomeLength));     %Avg fittness
    traceInfo(gen,4)=std(startPop(:,xZomeLength));
  end
  if ((abs(bval - oval)>epsilon) | (gen==1)) %If we have a new best sol
    if display
       fprintf(1,'\nGenerasi ke = %d\nNilai Fungsi Evaluasi = %f\n',...
       gen,bval);                                    %Update the display
    end
    if floatGA
      bPop(bFoundIn,:)=[gen startPop(bindx,:)]; %Update bPop Matrix
    else
      bPop(bFoundIn,:)=[gen b2f(startPop(bindx,1:numVar),....
         bounds,bits) startPop(bindx,xZomeLength)];
    end
    bFoundIn=bFoundIn+1;                      %Update number of changes
    oval=bval;                                %Update the best val
  else
    if display
      fprintf(1,'Generasi ke = %d ',gen);%Otherwise just update num gen
    end
  end
  endPop = feval(selectFN,startPop,[gen selectOps]); %Select

  if floatGA%Running with the model where the parameters are numbers of
           %ops
   for i=1:numXOvers,
    for j=1:xOverOps(i,1),
    a = round(rand*(popSize-1)+1); %Pick a parent
     b = round(rand*(popSize-1)+1); %Pick another parent
    xN=deblank(xOverFNs(i,:));  %Get the name of crossover function
     [c1 c2] = feval(xN,endPop(a,:),endPop(b,:),bounds,[gen....
                 xOverOps(i,:)]);
    if c1(1:numVar)==endPop(a,(1:numVar)) %Make sure we created a new
      c1(xZomeLength)=endPop(a,xZomeLength);%solution before evaluating
    elseif c1(1:numVar)==endPop(b,(1:numVar))
```

bersambung...

```
      c1(xZomeLength)=endPop(b,xZomeLength);
    else
      %[c1(xZomeLength) c1] = feval(evalFN,c1,[gen evalOps]);
      eval(e1str);
    end
    if c2(1:numVar)==endPop(a,(1:numVar))
      c2(xZomeLength)=endPop(a,xZomeLength);
    elseif c2(1:numVar)==endPop(b,(1:numVar));
      c2(xZomeLength)=endPop(b,xZomeLength);
    else
      %[c2(xZomeLength) c2] = feval(evalFN,c2,[gen evalOps]);
      eval(e2str);
    end
    endPop(a,:)=c1;
    endPop(b,:)=c2;
   end
  end
  for i=1:numMuts,
   for j=1:mutOps(i,1),
    a = round(rand*(popSize-1)+1);
    c1 = feval(deblank(mutFNs(i,:)),endPop(a,:),bounds,[gen....
         mutOps(i,:)]);
    if c1(1:numVar)==endPop(a,(1:numVar))
      c1(xZomeLength)=endPop(a,xZomeLength);
    else
      %[c1(xZomeLength) c1] = feval(evalFN,c1,[gen evalOps]);
      eval(e1str);
    end
    endPop(a,:)=c1;
   end
  end
else %We are running a probabilistic model of genetic operators
    for i=1:numXOvers,
      xN=deblank(xOverFNs(i,:));  %Get the name of crossover function
      cp=find(rand(popSize,1)<xOverOps(i,1)==1);
      if rem(size(cp,1),2) cp=cp(1:(size(cp,1)-1)); end
      cp=reshape(cp,size(cp,1)/2,2);
      for j=1:size(cp,1)
        a=cp(j,1); b=cp(j,2);
        [endPop(a,:) endPop(b,:)]=feval(xN,endPop(a,:),endPop(b,:),...
           bounds,[gen xOverOps(i,:)]);
      end
    end
    for i=1:numMuts
      mN=deblank(mutFNs(i,:));
      for j=1:popSize
       endPop(j,:) = feval(mN,endPop(j,:),bounds,[gen mutOps(i,:)]);
       eval(e1str);
      end
    end
  end
```

bersambung...

```
  gen=gen+1;
  done=feval(termFN,[gen termOps],bPop,endPop); %See if the ga is done
  startPop=endPop;                %Swap the populations
  [bval,bindx] = min(startPop(:,xZomeLength)); %Keep the best solution
  startPop(bindx,:) = best;             %replace it with the worst
end
[bval,bindx] = max(startPop(:,xZomeLength));
if display
       fprintf(1,'\nGenerasi ke = %d\nNilai Fungsi Evaluasi = %f\n',...
       gen,bval);%Update the display
end
%if display
%  fprintf(1,'\n%d %f\n',gen,bval);
%end
x=startPop(bindx,:);
if opts(2)==0 %binary
  x=b2f(x,bounds,bits);
  bPop(bFoundIn,:)=[gen b2f(startPop(bindx,1:numVar),bounds,bits)...
      startPop(bindx,xZomeLength)];
else
  bPop(bFoundIn,:)=[gen startPop(bindx,:)];
end
if collectTrace
  traceInfo(gen,1)=gen;           %current generation
  traceInfo(gen,2)=startPop(bindx,xZomeLength); %Best fittness
  traceInfo(gen,3)=mean(startPop(:,xZomeLength)); %Avg fittness
end
```

9. Initializega.m

```
function [pop] = initializega(num, bounds, evalFN,evalOps,options)
if nargin<5
  options=[1e-6 1];
end
if nargin<4
  evalOps=[];
end
if any(evalFN<48) %Not a .m file
 if options(2)==1 %Float GA
   estr=['x=pop(i,1); pop(i,xZomeLength)=', evalFN ';'];
 else %Binary GA
   estr=['x=b2f(pop(i,:),bounds,bits);pop(i,xZomeLength)=',evalFN ';'];
 end
else %A .m file
if options(2)==1 %Float GA
   estr=['[ pop(i,:)pop(i,xZomeLength)]='evalFN'(pop(i,:),[0 evalOps]);'];
 else %Binary GA
   estr=['x=b2f(pop(i,:),bounds,bits);[x v]=' evalFN ...
    '(x,[0 evalOps]); pop(i,:)=[f2b(x,bounds,bits) v];'];
    end
end
```

bersambung...

```
numVars      = size(bounds,1);            %Number of variables
rng          = (bounds(:,2)-bounds(:,1))'; %The variable ranges'
if options(2)==1 %Float GA
  xZomeLength = numVars+1;         %Length of string is numVar + fit
  pop         = zeros(num,xZomeLength);       %Allocate the new
population
  pop(:,1:numVars)=(ones(num,1)*rng).*(rand(num,numVars))+...
    (ones(num,1)*bounds(:,1)');
else %Binary GA
  bits=calcbits(bounds,options(1));
  xZomeLength = sum(bits)+1;       %Length of string is numVar + fit
  pop = round(rand(num,sum(bits)+1));
end
for i=1:num
  eval(estr);
end
```

10. Parse.m

```
function [x] = parse(inStr)
sz=size(inStr);
strLen=sz(2);
x=blanks(strLen);
wordCount=1;
last=0;
for i=1:strLen,
  if inStr(i) == ' '
    wordCount = wordCount + 1;
    x(wordCount,:)=blanks(strLen);
    last=i;
  else
    x(wordCount,i-last)=inStr(i);
  end
end
```

Lampiran B. Hasil Eksekusi pemrograman genetik untuk pemecahan masalah program linear

Jawaban Soal 1

Data Input :
Populasi : 3
Generasi : 5
Option :
-SelectFn : 0,04
-Crossover :
-arithXover = [1 0]
-heuristicXover = [1 3]
-simpleXover = [1 0]
-Mutasi :
-Boundary Mutasi = [2 0 0]
-unifmutasi = [2 0 0]

selectFn = roulette

startPop =

7,1769	8,5224
7,3329	8,4779
7,7016	8,3726

Generasi ke = 1
Nilai Fungsi Evaluasi terbaik = 8,522392
Generasi ke = 2
Nilai Fungsi Evaluasi terbaik = 8,855292
Generasi ke = 3
Generasi ke = 4
Nilai Fungsi Evaluasi terbaik = 9,001150
Generasi ke = 5
Nilai Fungsi Evaluasi Terbaik = 9,001150

endPop =

5,5000	9,0012
8,2711	8,2100
5,5000	9,0012

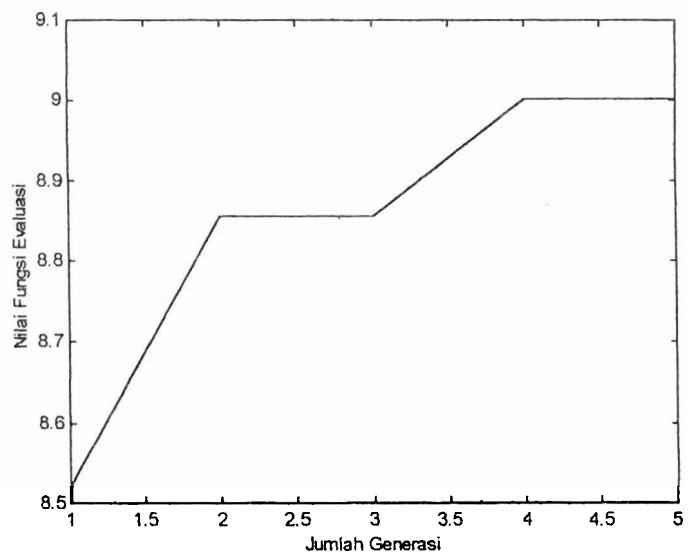
bestPop =

1	7,1769	8,5224
2	6,0109	8,8553
4	5,5000	9,0012
5	5,5000	9,0012

Nilai Variabel sol(1) = 5,5000
Nilai Fungsi Evaluasi = 9,0012
Nilai Variabel x_1 = 1,5003
Nilai Variabel x_2 = 1,0001
Variabel lain adalah : 0 (NOL)
Beban Komputasi = 639

trace =

1	8,5224	8,4576	0,0769
2	8,8553	8,6031	0,2230
3	8,8553	8,4896	0,3615
4	9,0012	8,8019	0,2306
5	9,0012	9,0012	0



Jawaban Soal 2

Data Input :

Populasi : 3
Generasi : 5
Option :
-SelectFn : 0,04
-Crossover :
-arithXover = [1 0]
-heuristicXover = [1 3]
-simpleXover = [1 0]
-Mutasi :
-Boundary Mutasi = [2 0 0]
-unifmutasi = [2 0 0]

selectFn = roulette

startPop =

1,0e+004 *

0,0042	0,0005	3,5230
0,0045	0,0006	4,1075
0,0050	0,0006	4,0290

Generasi ke = 1
Nilai Fungsi Evaluasi terbaik =41074,7208
Generasi ke = 2
Generasi ke = 3
Nilai Fungsi Evaluasi terbaik = 43495,669
Generasi ke = 4
Nilai Fungsi Evaluasi terbaik = 49965,034
Generasi ke = 5
Nilai Fungsi Evaluasi terbaik = 50500,000

endPop =

1,0e+004 *

0,0055	0,0006	3,7601
0,0045	0,0006	4,3496
0,0017	0,0006	5,0500

bestPop =

1,0e+004 *

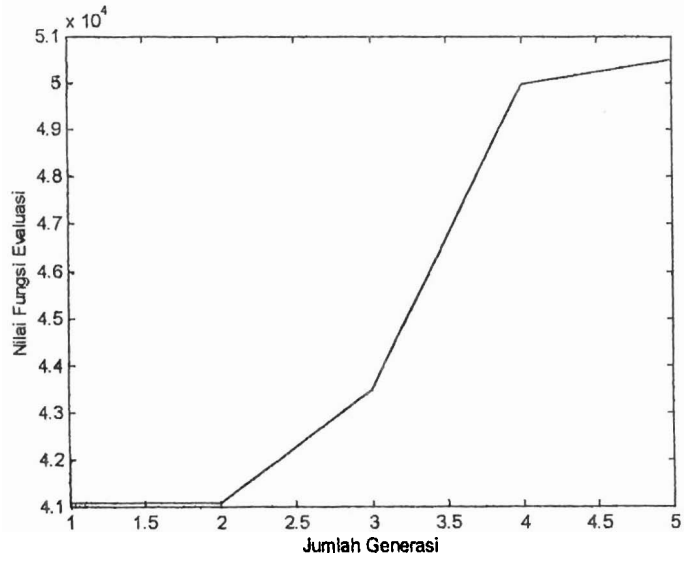
1	0,0045	0,0006	4,1075
3	0,0045	0,0006	4,3496
4	0,0017	0,0006	4,9965
5	0,0017	0,0006	5,0500

Nilai Variabel sol(1) = 16,5000
Nilai Variabel sol(2) = 6,5000
Nilai Fungsi Evaluasi =50500,0000
Nilai Variabel x_1 = 4,5000
Nilai Variabel x_2 = 7,0000
Variabel lain adalah : 0 (NOL)
Beban Komputasi = 727

trace =

1,0e+004 *

1	4,1075	3,8865	0,3172
2	4,1075	3,9587	0,2576
3	4,3496	4,2201	0,1219
4	4,9965	4,5652	0,3735
5	5,0500	4,7987	0



Jawaban Soal 3

Data Input :

Populasi : 3
Generasi : 5
Option :
-SelectFn : 0,04
-Crossover :
-arithXover = [1 0]
-heuristicXover = [1 3]
-simpleXover = [1 0]
-Mutasi :
-Boundary Mutasi = [2 0 0]
-unifmutasi = [2 0 0]

selectFn = roulette

startPop =

20,7617	61,0762
22,6931	61,2693
27,2582	61,7258

Generasi ke = 1
Nilai Fungsi Evaluasi terbaik = 61,725821
Generasi ke = 2
Generasi ke = 3
Nilai Fungsi Evaluasi terbaik = 63,333330
Generasi ke = 4
Generasi ke = 5

Nilai Fungsi Evaluasi terbaik = 63,333330

endPop =

0	59,0000
34,3085	62,4309
43,3333	63,3333

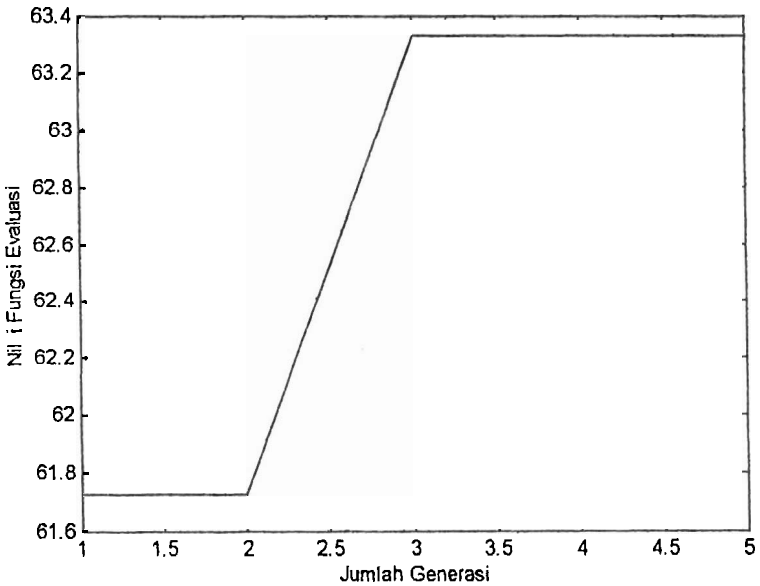
bestPop =

1	27,2582	61,7258
3	43,3333	63,3333
5	43,3333	63,3333

Nilai Variabel sol(1) = 43,3333
Nilai Fungsi Evaluasi = 63,3333
Nilai Variabel x_1 = 0,0000
Nilai Variabel x_2 = 3,3333
Nilai Variabel x_3 = 23,3333
Variabel lain adalah : 0 (NOL)
Beban Komputasi = 638.

trace =

1	61,7258	61,3571	0,3336
2	61,7258	61,6404	0,1480
3	63,3333	62,6087	0,8154
4	63,3333	62,2436	1,1987
5	63,3333	63,0325	0



Jawaban Soal 4

Data Input :
Populasi : 3
Generasi : 5
Option :
-SelectFn : 0,04
-Crossover :
-arithXover = [1 0]
-heuristicXover = [1 3]
-simpleXover = [1 0]
-Mutasi :
-Boundary Mutasi = [2 0 0]
-unifmutasi = [2 0 0]

selectFn = roulette

startPop =

10,5406	-14,5942
11,5211	-4,7886
13,8388	18,3879

Generasi ke = 1
Nilai Fungsi Evaluasi terbaik = 18,387925
Generasi ke = 2
Generasi ke = 3
Nilai Fungsi Evaluasi terbaik = 100,00000
Generasi ke = 4
Generasi ke = 5

Nilai Fungsi Evaluasi terbaik = 100,00000

endPop =

0	-120,0000
17,4182	54,1819
22,0000	100,0000

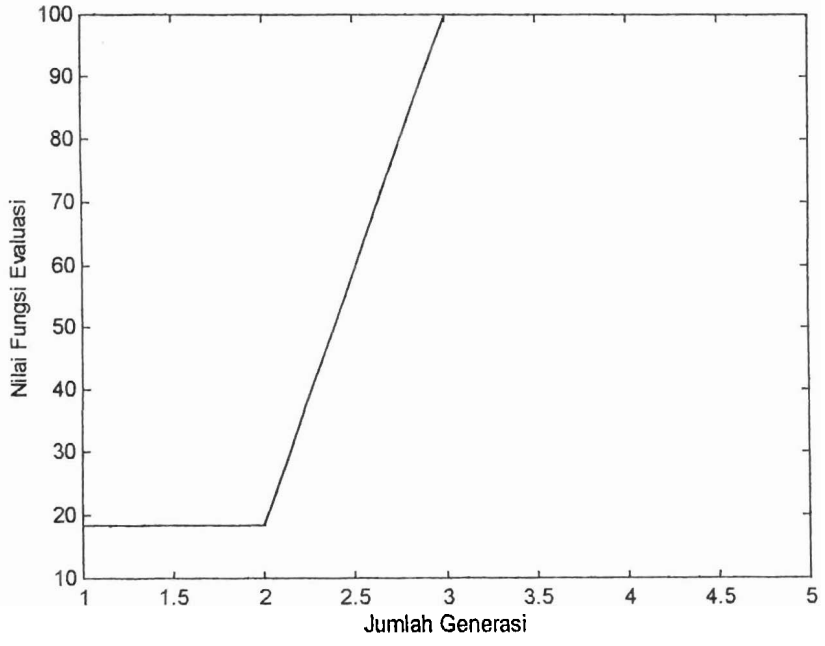
bestPop =

1	13,8388	18,3879
3	22,0000	100,0000
5	22,0000	100,0000

Nilai Variabel sol(1) = 22,0000
Nilai Fungsi Evaluasi =100,0000
Nilai Variabel x_1 = 10,0000
Nilai Variabel x_2 = 0,0000
Nilai Variabel x_3 = 20,0000
Variabel lain adalah : 0 (NOL)
Beban Komputasi = 628

trace =

1	18,3879	- 0,3316	16,9368
2	18,3879	3,0564	16,6129
3	100,0000	63,2129	41,3955
4	100,0000	50,2457	62,9428
5	100,0000	84,7273	0



Jawaban Soal 5

Data Input :

Populasi : 3
Generasi : 5
Option :
-SelectFn : 0,04
-Crossover :
-arithXover = [1 0]
-heuristicXover = [1 3]
-simpleXover = [1 0]
-Mutasi :
-Boundary Mutasi = [2 0 0]
-unifmutasi = [2 0 0]

selectFn = roulette

```
startPop =
        1,0e+003 *
         0,0067   1,4282
         0,0069   1,4215
         0,0073   1,4057
```

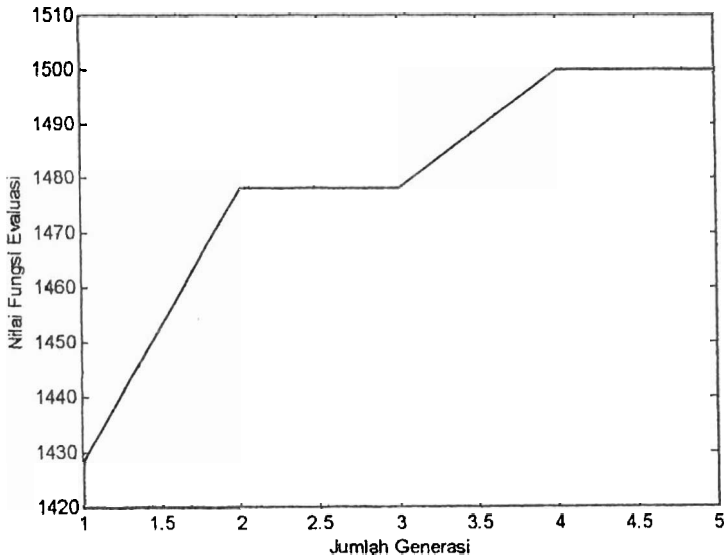
Generasi ke = 1
Nilai Fungsi Evaluasi terbaik = 1428,1546
Generasi ke = 2
Nilai Fungsi Evaluasi terbaik = 1478,122
Generasi ke = 3
Generasi ke = 4
Nilai Fungsi Evaluasi terbaik = 1500,015
Generasi ke = 5
Nilai Fungsi Evaluasi terbaik = 1500,015

```
endPop =
        1,0e+003 *
         0,0045   1,5000
         0,0081   1,3813
         0,0045   1,5000
```

```
bestPop =
            1,0e+003 *
        1   0,0067   1,4282
        2   0,0052   1,4781
        4   0,0045   1,5000
        5   0,0045   1,5000
```

Nilai Variabel sol(1) = 4,5000
Nilai Fungsi Evaluasi =1500,0150
Nilai Variabel x_1 = 2,5000
Nilai Variabel x_2 = -0,0001
Nilai Variabel x_3 = 12,4999
Variabel lain adalah : 0 (NOL)
Beban Komputasi = 639

```
trace =
            1,0e+003 *
        1   1,4282   1,4184   0,0115
        2   1,4781   1,4403   0,0335
        3   1,4781   1,4232   0,0543
        4   1,5000   1,4701   0,0346
        5   1,5000   1,5000        0
```



Jawaban Soal 6

Data Input :

Populasi : 3
Generasi : 5
Option :
-SelectFn : 0,04
-Crossover :
-arithXover = [1 0]
-heuristicXover = [1 3]
-simpleXover = [1 0]
-Mutasi :
-Boundary Mutasi = [2 0 0]
-unifmutasi = [2 0 0]

selectFn = roulette

startPop =

0,7725	21,2275
0,8134	21,1866
0,9100	21,0900

Generasi ke = 1
Nilai Fungsi Evaluasi terbaik = 21,227467
Generasi ke = 2
Nilai Fungsi Evaluasi terbaik = 21,532849
Generasi ke = 3
Generasi ke = 4
Nilai Fungsi Evaluasi terbaik = 21,666650
Generasi ke = 5
Nilai Fungsi Evaluasi terbaik = 21,666650

endPop =

0,3333	21,6667
1,0591	20,9409
0,3333	21,6667

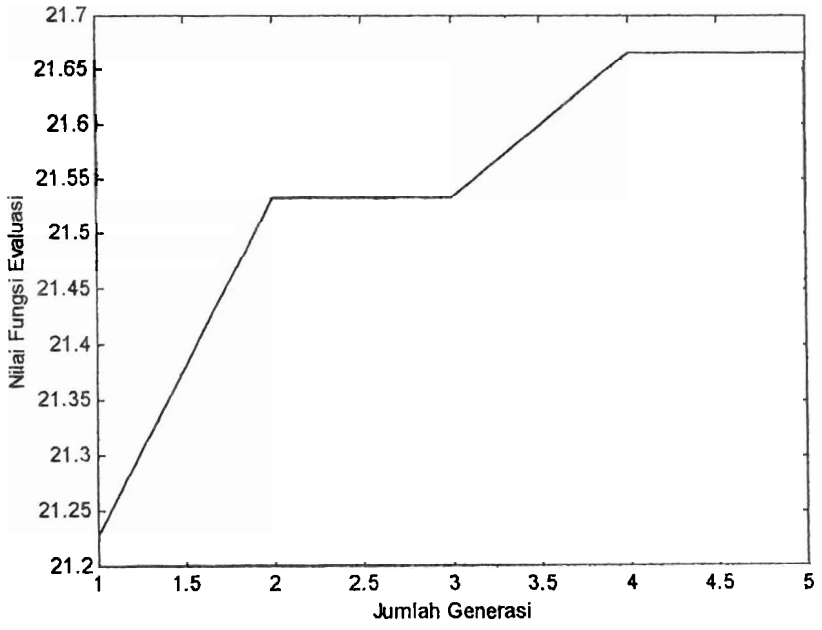
bestPop =

1	0,7725	21,2275
2	0,4672	21,5328
4	0,3333	21,6667
5	0,3333	21,6667

Nilai Variabel sol(1) = 0,3333
Nilai Fungsi Evaluasi = 21,6667
Nilai Variabel x_1 = 3,6667
Nilai Variabel x_2 = -0,0000
Variabel lain adalah : 0 (NOL)
Beban Komputasi = 620

trace =

1	21,2275	21,1680	0,0706
2	21,5328	21,3015	0,2046
3	21,5328	21,1974	0,3316
4	21,6667	21,4839	0,2116
5	21,6667	21,6667	0



Jawaban Soal 7

Data Input :

Populasi : 3
Generasi : 5
Option :
-SelectFn : 0,04
-Crossover :
-arithXover = [1 0]
-heuristicXover = [1 3]
-simpleXover = [1 0]
-Mutasi :
-Boundary Mutasi = [2 0 0]
-unifmutasi = [2 0 0]

selectFn = roulette

startPop =

1,9165	0,8339	22,7504
2,0948	3,1351	25,2298
2,5161	3,3580	25,8741

Generasi ke = 1
Nilai Fungsi Evaluasi terbaik = 25,874138
Generasi ke = 2
Nilai Fungsi Evaluasi terbaik = 27,816986
Generasi ke = 3
Nilai Fungsi Evaluasi terbaik = 27,971445
Generasi ke = 4
Nilai Fungsi Evaluasi terbaik = 28,000000
Generasi ke = 5
Nilai Fungsi Evaluasi terbaik = 28,000000

endPop =

2,8669	2,8164	25,6833
4,0000	4,0000	28,0000
0	4,0000	24,0000

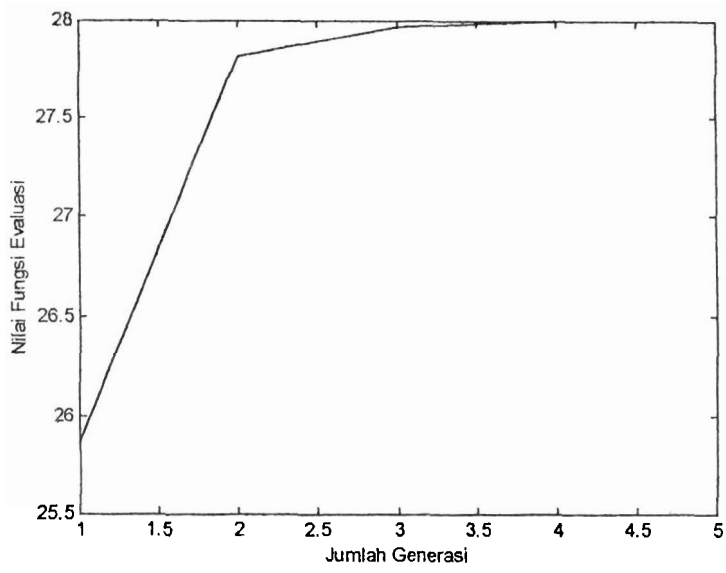
bestPop =

1	2,5161	3,3580	25,8741
2	4,0000	3,8170	27,8170
3	4,0000	3,9714	27,9714
4	4,0000	4,0000	28,0000
5	4,0000	4,0000	28,0000

Nilai Variabel sol(1) = 4,0000
Nilai Variabel sol(2) = 4,0000
Nilai Fungsi Evaluasi = 28,0000
Nilai Variabel x_1 = 0,0003
Nilai Variabel x_2 = 0,0003
Variabel lain adalah : 0 (NOL)
Beban Komputasi = 698

trace =

1	25,8741	24,6181	1,6493
2	27,8170	26,3070	1,3468
3	27,9714	27,4349	0,7994
4	28,0000	27,4959	0,8486
5	28,0000	27,2278	0



Lampiran C. Listing program fungsi barnes dan hasil eksekusi

1. Listing Program Fungsi Barnes

```
function [xsol,basic]=barnes(A,b,c,tol);
% Algorithma Barnes untuk Pemrograman Linier;
% FUNCTION [xsol,basic]=barnes(A,b,c,tol);
xZ=[]; x=[]; [m,n]=size(A);
aplus1=b-sum(A(1:m,:)')'; cplus1=1000000;
A=[A aplus1]; c=[c cplus1];
B=[ ]; n=n+1;
x0=ones(1,n)';x=x0;
alpha=0.0001; lambda=zeros(1,m)'; iter=0;
%main step
while abs(c*x-lambda'*b)>tol
   xZ=x.*x; D=diag(x); DZ=diag(xZ);
   ADZ=A*DZ;
   lambda=(ADZ*A')\(ADZ*c');
   dualres =c'-A'*lambda;
   normres=norm(D*dualres);
   for i=1:n
     if dualres(i)>0
        ratio(i)=normres/(x(i)*(c(i)-A(:,i)'*lambda));
     else
       ratio(i)=inf;
     end
   end
   R=min(ratio)-alpha;
   x1=x-R*DZ*dualres/normres;
   x=x1; basiscount=0; B=[ ]; basic=[ ]; cb = [ ];
   for k=1:n
     if x(k)>tol
       basiscount=basiscount+1;
       basic=[basic k];
     end
   end
   %only used if problem non=degenerate
   if basiscount==m
     for k=basic
       B=[B A(:,k)]; cb=[cb c(k)];
     end
     primalsol=b'/B'; xsol=primalsol;
     break
   end
   iter=iter+1;
end;
objective=c*x
xsol=x;
```

2. Soal Program Linear

```
%Pemecahan soal program linier dan transportasi linear
clear;
clc;
flops(0);
%soal 1
%c=[-4 -3 0 0 0];
%a=[2 3 1 0 0;-3 2 0 1 0;2 1 0 0 1];
%b=[6;3;4];
%soal 2
%c=[-5000 -4000 0 0 0 0 0];
%a=[10 15 1 0 0 0 0;20 10 0 1 0 0 0;30 10 0 0 -1 0 0;1 -3 0 0 0 1 0;....
    1 1 0 0 0 0 -1];
%b=[150;160;135;0;5];
%soal 3
%c=[-1 2 -3 0 0];
%a=[1 1 1 1 0 ;1 -1 1 0 -1;3 -1 -1 0 0];
%b=[70;20;-50];
%soal 4
%c=[-2 4 -4 0 0 ];
%a=[1 2 1 1 0;1 1 1 0 -1;1 1 0 0 0];
%b=[30;8;10];
%soal 5
%c=[100 100 100 0 0 0];
%a=[2 2 1 1 0 0;2 1 2 0 -1 0;0 1 2 0 0 -1];
%b=[22;30;25];
%soal 6
%c=[5 7 10 0 0];
%a=[1 1 1 -1 0;1 2 4 0 -1];
%b=[4;5];
%soal 7
%c=[-1 -2 -3 -4 0 0];
%a=[1 2 2 3 1 0;2 1 3 2 0 1];
%b=[20;20];

flops(0);
[xsol,ind]=barnes(a,b,c,0.000005);
fff=flops;
i=1;fprintf('\nPemecahannya adalah :');
 for i=ind
     fprintf('\nx(%1.0f)=%8.4f\',i,xsol(i));
     i=i+1;
 end
 fprintf('\nVariabel lain adalah : 0 (NOL)\n');
 disp(['Flops = ',num2str(fff)]);
```

3. Hasil Eksekusi Program Fungsi Barnes.

Jawaban Soal 1.
Nilai Fungsi objektif = 9,0000
Nilai Variabel x_1 = 1,5000
x_2 = 1,0000
Beban Komputasi (flops) = 5262

Jawaban Soal 2.
Nilai Fungsi objektif = 50.5000
Nilai Variabel x_1 = 4,5000
x_2 = 7,0000
Beban Komputasi (flops) = 9771

Jawaban Soal 3.
Nilai Fungsi objektif = 63,3333
Nilai Variabel x_1 = 0,0000
x_2 = 3,3333
x_3 = 23,3333
Beban Komputasi (flops) = 5281

Jawaban Soal 4.
Nilai Fungsi objektif = 100,0000
Nilai Variabel x_1 = 10,0000
x_2 = 0,0000
x_3 = 20,3333
Beban Komputasi (flops) = 4387

Jawaban Soal 5.
Nilai Fungsi objektif = 1500,0000
Nilai Variabel x_1 = 2,5000
x_2 = 0,0000
x_3 = 12,5000
Beban Komputasi (flops) = 7468

Jawaban Soal 6.
Nilai Fungsi objektif = 21,6667
Nilai Variabel x_1 = 3,6667
x_2 = 0,0000
x_3 = 0,3333
Beban Komputasi (flops) = 4938

Jawaban Soal 7.

Nilai Fungsi objektif = 28,0000
Nilai Variabel x_1 = 0,0000
x_2 = 0,0000
x_3 = 4,0000
x_4 = 4,0000
Beban Komputasi (flops) = 6278