PERANCANGAN DAN PEMBUATAN PERANGKAT LUNAK UNTUK PENJADUALAN JOB SHOP DENGAN MENGGUNAKAN ALGORITMA GENETIK

TUGAS AKHIR

Oleh :

ERIE KRESNA ANDANA

NRP. 2693.100.017

JURUSAN TEKNIK INFORMATIKA
FAKULTAS TEKNOLOGI INDUSTRI
INSTITUT TEKNOLOGI SEPULUH NOPEMBER
SURABAYA

2001

PERANCANGAN DAN PEMBUATAN PERANGKAT LUNAK UNTUK PENJADUALAN JOB SHOP DENGAN MENGGUNAKAN ALGORITMA GENETIK

TUGAS AKHIR

Diajukan Guna Memenuhi Sebagian Persyaratan
Untuk Memperoleh Gelar Sarjana Komputer
Pada
Jurusan Teknik Informatika
Fakultas Teknologi Industri
Institut Teknologi Sepuluh Nopember
Surabaya

Mengetahui / Menyetujui,

Dosen Pembimbing I	Dosen Pembimbing II
Drs.Ec. Ir. Riyanarto Sarno, M.Sc., Ph.D.	**Yudhi Purwananto, S.Kom**

SURABAYA
Agustus, 2001

" Bacalah dengan (menyebut) nama Tuhanmu yang menciptakan

Dia telah menciptakan manusia dari segumpal darah

Bacalah, dan Tuhanmu itu amat mulia

Yang mengajar (manusia) dengan kalam

Dia mengajarkan kepada manusia apa yang tidak diketahuinya

..... "

(Q.S. Al - 'Alaq : 1-5)

" ..... niscaya Allah akan meninggikan orang - orang yang beriman di antaramu

dan orang - orang yang diberi ilmu pengetahuan beberapa derajat ..... "

(Q.S. Al-Mujadilah : 11)

ABSTRAK

Sistem produksi yang melibatkan banyak proses, mesin dan juga waktu proses yang bervariasi, membutuhkan penjadualan yang tepat. Salah satu tipe penjadualan yang digunakan adalah *job shop*. Penjadualan *job shop* memiliki bermacam jenis dan dapat diselesaikan dengan beberapa metode. Tujuan penjadualan adalah menentukan urutan pengerjaan *job* pada mesin yang disediakan dengan waktu minimal, yang disebut *makespan*. Pada tugas akhir ini digunakan algoritma genetik untuk menyelesaikan permasalahan yang hanya dibatasi pada penjadualan *job shop* klasik dengan pola kedatangan *job* statis. Sebagai pembanding digunakan metode heuristik dengan menggunakan aturan prioritas (*priority dispatching rule*), dalam hal ini *shortest processing time* (SPT). Hasil algoritma genetik yang diharapkan adalah diperoleh *makespan* lebih kecil dalam waktu yang lebih cepat dari pada menggunakan SPT.

Algoritma genetik yang dibuat menggunakan representasi permutasi *job* yang disebut dengan *job based representation*. Operator tukar silang menggunakan *order* dan *position based crossover*. Sedangkan operator mutasi menggunakan *reciprocal exchange* dan *insertion*. Metode seleksi menggunakan *roulette wheel* dan *elitism*. Untuk memperoleh parameter algoritma genetik yang dapat menghasilkan *makespan* minimal, uji coba dilakukan dalam beberapa dimensi permasalahan, yaitu dengan memberikan data jumlah *job* dan mesin serta nilai ukuran populasi, tingkat tukar silang, dan tingkat mutasi yang berbeda – beda. Dari beberapa parameter algoritma genetik tersebut dapat ditemukan nilai parameter yang dapat menghasilkan *makespan* minimal. Kemudian dengan jumlah *job* dan mesin yang sama pada uji coba algoritma genetik itu dilakukan uji coba pada metode SPT.

Hasil yang diperoleh adalah algoritma genetik dapat menghasilkan *makespan* lebih kecil dengan waktu lebih cepat dari pada SPT. Selain itu juga dikaji cara untuk memperkirakan parameter algoritma genetik.

KATA PENGANTAR

Alhamdulillaah, segala puja dan puji syukur hanya kepada Allah SWT yang telah memberikan rahmat dan hidayah-Nya, sehingga setiap yang dikerjakan makhluk-Nya selalu diberikan kemudahan untuk menyelesaikannya. Shalawat dan salam semoga tercurahkan kepada Rasulullah, junjungan kita Nabi besar Muhammad SAW. dan keluarganya. Dan seluruh pengikutnya yang memegang teguh ajaran yang disampaikan beliau agar diberi limpahan kasih sayang dan keselamatan dari-Nya.

Penyusun mengucapkan rasa syukur ke hadhirat Allah SWT dengan diberinya kekuatan dan kesempatan untuk menyelesaikan penyusunan tugas akhir ini, dengan judul : **"Perancangan dan Pembuatan Perangkat Lunak untuk Penjadualan Job Shop dengan Menggunakan Algoritma Genetik"**. Tugas akhir ini diajukan untuk memenuhi sebagian persyaratan untuk memperoleh gelar sarjana komputer di Jurusan Teknik Informatika, Institut Teknologi Sepuluh Nopember Surabaya. Semoga tugas akhir ini bermanfaat bagi semua pihak, khususnya para pembaca.

Dan selanjutnya penyusun sangat menyadari bahwa masih ada kekurangan yang terdapat pada penyusunan tugas akhir ini. Untuk itu penyusun mengharapkan banyak masukan dari para pembaca agar dapat dilakukan penyempurnaan melalui penelitian lebih lanjut.

Surabaya, Agustus 2001

Penyusun

UCAPAN TERIMA KASIH

Dengan selesainya penyusunan tugas akhir ini, maka dalam kesempatan ini penyusun ingin mengucapkan terima kasih sebanyak – banyaknya kepada :

1. Bapak Ir. Arif Djunaidy, M.Sc., Phd. sebagai Ketua Jurusan Teknik Informatika, Institut Teknologi Sepuluh Nopember Surabaya
2. Ibu Dr. Ir. Handayani Tjandrasa, M.Sc. sebagai dosen wali saya yang telah memberikan banyak motivasi saya selama masa kuliah
3. Bapak Drs.Ec. Ir. Riyanarto Sarno, M.Sc., Phd. sebagai Dosen Pembimbing I yang telah banyak memberikan saran dan membagikan ilmu pengetahuan dan wawasannya kepada penyusun
4. Bapak Yudhi Purwananto, S.Kom. sebagai Dosen Pembimbing II yang telah banyak membagikan ilmunya kepada saya
5. Bapak dan Ibu Dosen di Jurusan Teknik Informatika - ITS yang telah memberikan materi kuliah, ilmu pengetahuan, wawasan, dan nasihat yang sangat berguna sebagai *trigger* bagi mahasiswanya
6. Papa dan mama saya, Bapak Subardjo, S.E. dan Ibu Dra. Ec. Dyah Trisnowati yang telah tulus dan ikhlas membesarkan dan membimbing saya sejak sebelum dilahirkan sampai mendapatkan arah untuk melangkah ke masa depan. Semoga apa yang telah diberikan dapat bernilai sebagai amalan yang mendapatkan pahala terus – menerus.

7. Keluarga besar dari kakek-nenek, yaitu keluarga besar Mbah Mochamad Icksan dan Mbah Wignjowijoto yang telah memberikan teladan kejujuran dan kerja keras pantang menyerah
8. Istri tersayang, adinda Erlina Mis Sulistyowati, S.H. yang selalu menemani dalam segala keadaan baik dalam nikmat maupun cobaan dan ujian. Semoga segalanya dalam ridha Allah SWT
9. Bapak dan Ibu Guru serta Ustadz dan Ustadzah yang telah mendidik, mengajar, dan menanamkan dasar ilmu pengetahuan dan akhlaq kepada saya
10. Seluruh staf Jurusan Teknik Informatika – ITS yang sangat bersahabat dan memudahkan segala urusan mahasiswa, seperti Mas Yudi, Pak Mu'in, Mas Sugeng, Mbak Irna, Pak Sholeh, petugas RBC yaitu Mbak Davi dan Mas Kadir, Pak Satpam yang penuh dedikasi menjaga kendaraan kami, dan staf lain yang mungkin belum saya sebutkan
11. Saudara - saudaraku satu angkatan C-09 yang saling memotivasi, menolong, dan memudahkan urusan temannya yang lain sejak 'masa – masa sulit' sebelum kuliah sampai pada 'masa – masa sulit' menyelesaikan TA
12. Rekan – rekan seperjuangan yang saling bahu – membahu dalam menghadapi ujian TA, khususnya dari C-09 yang tersisa
13. Para S.Kom dan TA-wan tentang GA seperti Mas Anang, Mas Edi, Mas Anto', Mas Koko', dan Mbak Dewi
14. Mbak Entin dan Mbak Umi yang sangat membantu saya saat program saya mengalami kebuntuan

15. Mas Sony dan Mas B. Heriyanto yang secara tidak langsung memberi inspirasi judul TA saya sewaktu saya membaca TA-nya
16. Saudara – saudara pemilik Takafuli Computer dan rekan – rekannya. Spesial buat pinjaman lunak sekali berupa *printer*, CD ROM, tinta suntikan, nge*print* dan ambil kertas rim bayar belakang, dan lain - lain
17. Seluruh pihak yang belum saya sebutkan yang telah sangat membantu dalam penyelesaian tugas akhir saya.

Semoga tugas akhir ini bermanfaat bagi semua dan semoga segala perhatian dan bantuannya dinilai oleh Allah SWT sebagai amalan baik yang mendapatkan pahala yang berlipat – lipat. Amin.

Penyusun

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

Halaman

BAB I

PENDAHULUAN

I.1. Latar belakang

Dalam sebuah sistem produksi yang kompleks dapat terjadi penumpukan pekerjaan atau barang yang membentuk antrian panjang yang tidak dapat diselesaikan dengan optimal. Sistem yang melibatkan banyak proses, mesin dan juga waktu proses yang bervariasi, akan menemui banyak hambatan bila tidak menggunakan metode penjadualan yang tepat. Dan pada akhirnya akan berakibat pada proses produksi secara keseluruhan. Sistem tidak dapat bekerja secara efektif dan efisien.

Oleh karena itu diperlukan suatu metode penjadualan. Penjadualan dapat dideskripsikan sebagai berikut : Ada n pekerjaan (*job*) yang harus diproses. Setiap *job* mempunyai waktu *setup*, waktu proses, dan batas waktu selesai dikerjakan. Untuk menyelesaikan setiap *job*, diberikan beberapa mesin untuk memprosesnya menurut urutan tertentu. Permasalahan penjadualan adalah menentukan urutan tersebut sedemikian hingga didapat hasil yang optimal [Elsayed dan Boucher, 1994].

Salah satu tipe penjadualan mesin yang sering digunakan adalah penjadualan *job shop*. Secara sederhana penjadualan *job shop* dapat diilustrasikan sebagai penjadualan banyak pekerjaan yang harus dikerjakan melalui banyak mesin. Setiap mesin hanya memproses sebuah pekerjaan setiap satuan waktu dan tidak akan memproses pekerjaan yang sama lebih dari sekali. Setiap mesin

memproses pekerjaan menurut aturan tertentu, sehingga diperbolehkan urutan pengerjaan untuk tiap mesin tidak sama. Sedangkan tujuan penjadualan adalah untuk mendapatkan jadual yang optimal, yaitu membentuk urutan sedemikian hingga waktu proses dapat ditekan sampai minimal yang disebut *makespan* [Cheng dan Gen, 1996]. Untuk membentuk jadual yang optimal dapat digunakan beberapa metode.

Metode tersebut pada umumnya dikategorikan dalam dua jenis, yaitu : metode analitis dan metode heuristik. Metode analitis adalah metode yang menyelesaikan masalah dengan matematis, sehingga dapat ditemukan solusi yang tepat untuk suatu permasalahan. Misalnya masalah penjadualan, jika diselesaikan dengan metode matematis, akan didapatkan solusi tepat yang merupakan jadual optimal dari seluruh kemungkinan jadual yang dapat dibentuk. Solusi ini disebut *global optima*. Namun metode ini memiliki kelemahan pada aspek komputasi. Semakin besar ukuran permasalahan, waktu yang dibutuhkan akan meningkat secara eksponensial [Mortton dan Pentico, 1993].

Untuk mengatasi kelemahan ini digunakan metode heuristik. Khusus untuk penjadualan *job shop*, karena bentuk permasalahannya merupakan kombinatorik NP kompleks, maka metode yang dapat digunakan untuk mempersingkat waktu adalah metode heuristik dengan menggunakan aturan prioritas (*priority dispatching rule*). Namun metode ini belum tentu mendapatkan solusi optimal yang tepat dan sering disebut bahwa metode ini hanya mencapai solusi *local optima*.

Selain kedua jenis metode tersebut terdapat cara pendekatan baru untuk menyelesaikan permasalahan, khususnya penjadualan *job shop*. Pendekatan tersebut menggunakan algoritma genetik. Algoritma genetik bekerja dengan meniru proses evolusi makhluk hidup. Di dalam algoritma ini banyak digunakan terminologi yang diadopsi dari istilah – istilah dalam ilmu genetika. Algoritma genetik berangkat dari himpunan solusi yang disebut kromosom. Kromosom terdiri atas gen. Untuk setiap kromosom dilakukan proses penilaian, dikombinasikan gen – gen yang terdapat di dalamnya dengan proses tukar silang (*crossover*) dan mutasi (*mutation*), sehingga terbentuk komosom baru. Pada kromosom baru dilakukan penilaian kembali untuk menentukan proses berikutnya yang sama dengan sebelumnya. Pada akhirnya algoritma diharapkan konvergen pada kromosom yang menunjukkan solusi permasalahan secara keseluruhan [Cheng dan Gen, 1996]. Bila digunakan dalam sistem penjadualan, algoritma genetik diharapkan mampu menghasilkan kromosom yang memuat solusi jadual yang optimal.

I.2. Tujuan

Tugas akhir ini bertujuan untuk merancang dan membuat perangkat lunak yang mengimplementasikan algoritma genetik untuk penjadualan *job shop*. Algoritma genetik juga merupakan metode heuristik. Dengan menggunakan algoritma genetik diharapkan penyelesaian permasalahan penjadualan *job shop* dapat mencapai tujuannya, yaitu menemukan jadual dengan *makespan* lebih kecil

dengan waktu yang lebih cepat dari metode heuristik. Metode heuristik yang digunakan sebagai pembanding adalah *shortest processing time* (SPT).

I.3. Batasan permasalahan

Secara umum pada penjadualan *job shop* dapat dibentuk banyak macam jadual, karena pada proses penjadualannya dapat disisipkan sebarang waktu *idle*. Oleh karena itu terdapat pula jenis – jenis jadual. Sehingga perlu dijelaskan pula penjadualan *job shop* yang akan dibuat pada tugas akhir ini.

Demikian pula dengan pola kedatangan pekerjaan yang diberikan kepada sistem mempunyai spesifikasi tersendiri. Pola kedatangan dapat dibagi menjadi dua, yaitu : statis dan dinamis [Elsayed dan Boucher, 1994].

Sedangkan algoritma genetik yang digunakan untuk menyelesaikan permasalahan penjadualan *job shop* telah banyak menjadi bahan penelitian. Terbukti dengan banyaknya cara dalam representasi solusi dalam kromosom. Demikian juga dengan metode tukar silang (*crossover*) dan mutasi (*mutation*) yang digunakan.

Algoritma genetik yang dibuat adalah mencoba menerapkan metode yang telah dilakukan oleh para peneliti tersebut. Pada penerapan tersebut, algoritma genetik yang digunakan tidak terbatas menggunakan referensi seorang peneliti secara utuh, melainkan mencoba mengambil dari beberapa referensi dan menggabungkannya. Misalnya, metode representasi dan tukar silang tidak selalu menggunakan referensi yang sama.

Oleh karena hal tersebut di atas, maka pada tugas akhir ini digunakan batasan masalah sebagai berikut :

- Sistem penjadualan *job shop* yang dipecahkan adalah *job shop* klasik
- Pola kedatangan *job* menggunakan pola kedatangan statis
- Jadual yang dibentuk adalah jenis jadual aktif
- Batasan waktu pada *job* dan mesin hanya menggunakan waktu proses, diasumsikan waktu tersebut termasuk waktu *setup* mesin dan tidak ada penundaan/keterlambatan penyelesaian *job*
- Metode representasi menggunakan *job based* dan *operation based representation*
- Metode tukar silang untuk menggunakan *order* dan *position based crossover* untuk *job based representation*. Sedangkan *order based* dan *precedence preservative crossover* digunakan untuk *operation based representation*
- Metode mutasi menggunakan *reciprocal exchange* dan *insertion* untuk *job based representation*. Dan *order based mutation* untuk *operation based representation*
- Metode seleksi menggunakan *roulette wheel* dengan *elitism*
- Penskalaan *fitness* (*fitness scale*) menggunakan *windowing* dan *exponential*.

I.4. Kontribusi

Tugas akhir ini bermanfaat untuk menemukan solusi penjadualan *job shop* dengan algoritma genetik dan dibandingkan dengan metode heuristik dalam hal menentukan jadual dengan *makespan* minimal dan waktu penjadualan. Dengan

algoritma genetik dapat menemukan *makespan* yang lebih kecil dengan waktu yang lebih cepat dari pada menggunakan metode SPT.

I.5. Metodologi

Penyusunan tugas akhir ini menggunakan beberapa tahapan langkah sebagai berikut :

1. Perumusan dan pemahaman masalah
2. Studi literatur
3. Perancangan perangkat lunak
4. Evaluasi awal terhadap rancangan perangkat lunak
5. Penyempurnaan perangkat lunak
6. Evaluasi akhir terhadap perangkat lunak
7. Pembuatan laporan.

I.6. Sistematika laporan

Laporan tugas akhir ini disusun menurut sistematika sebagai berikut :

Bab I Pendahuluan

Bab ini menjelaskan latar belakang, tujuan, dan batasan permasalahan yang dihadapi pada penyusunan tugas akhir ini. Dalam bab ini juga menjelaskan kontribusi yang diberikan dan metodologi yang digunakan dalam penyusunan tugas akhir ini serta sistematika penyusunan laporannya

Bab II Teori Penjadualan

Bab ini menjelaskan beberapa dasar teori tentang sistem penjadualan mesin dan lebih diarahkan pejelasannya kepada sistem penjadualan *job shop*. Pada awal bab ini akan diuraikan secara singkat penjadualan *flow shop* sebagai arahan untuk membantu memahami sistem penjadualan *job shop*

Bab III Algoritma genetik

Bab ini menjelaskan konsep dasar dan cara kerja algoritma genetik yang akan digunakan untuk merancang dan membuat perangkat lunak pada tugas akhir ini

Bab IV Penerapan algoritma genetik untuk penjadualan job shop

Bab ini menjelaskan penerapan algoritma genetik yang digunakan pada tugas akhir ini yang terdiri atas : representasi solusi, fungsi evaluasi, metode tukar silang, metode mutasi, dan seleksi

Bab V Perancangan dan implementasi perangkat lunak

Pada bab ini akan dijelaskan rancangan perangkat lunak sederhana yang akan dibuat sebagai solusi pada tugas akhir ini. Pada bagian implementasi akan menguraikan beberapa kelas atau obyek yang akan dibuat beserta penjelasannya dengan menggunakan bahasa pemrograman tertentu

Bab VI Uji coba dan analisa

Bab ini menjelaskan uji coba dan analisa algoritma genetik, yaitu cara mendapatkan parameter terbaik untuk beberapa contoh permasalahan yang diberikan dan membandingkannya dengan metode heuristik lain yang hasilnya dapat dilihat pada tampilan keluaran berupa grafik dan nilai *makespan*

Bab VII Kesimpulan dan saran

Bab ini menjelaskan beberapa hal sebagai kesimpulan yang didapat pada saat evaluasi terhadap kinerja algoritma yang digunakan. Pada bab ini juga disebutkan beberapa saran yang diperlukan untuk menyempurnakan hasil tugas akhir ini agar dapat dilakukan penelitian lebih lanjut pada masa mendatang.

BAB II

TEORI PENJADUALAN

Dalam sistem produksi yang kompleks, yang terdiri atas banyak pekerjaan dan mesin serta waktu proses yang bervariasi antara proses pengerjaan setiap pekerjaan, membutuhkan teknik penjadualan yang tepat. Semakin kompleks sistem produksi, maka teknik penjadualan harus semakin baik.

Bila sebuah sistem produksi harus mengerjakan beberapa pekerjaan (*job*) dengan menggunakan beberapa mesin dan urutan pengerjaan *job* pada setiap mesinnya adalah sama, maka sistem produksi tersebut dikatakan menggunakan penjadualan pola *flow shop*. Dan bila urutan pengerjaan *job* setiap mesinnya tidak sama, maka sistem tersebut berpola *job shop*.

Permasalahan penjadualan dapat dideskripsikan sebagai berikut : Ada beberapa buah pekerjaan dan beberapa buah mesin. Setiap pekerjaan terdiri dari beberapa operasi yang harus diselesaikan dengan mesin yang disediakan. Adanya metode tertentu untuk dapat membagi pekerjaan – pekerjaan tersebut kepada mesin – mesin dan menyusunnya dalam suatu urutan tertentu, sehingga seluruh pekerjaan diselesaikan dengan waktu minimal, yang disebut dengan *makespan*.

Sedangkan untuk pola kedatangan *job* yang harus diselesaikan terbagi menjadi dua jenis, yaitu : statis dan dinamis. Pola kedatangan dikatakan statis bila tidak ada kedatangan *job* baru saat sistem produksi sedang menyelesaikan *job* – *job* yang telah diberikan selama periode tertentu. Namun bila pada saat sistem sedang menyelesaikan *job* diberikan *job* baru, maka *job* tersebut dikatakan berpola

kedatangan dinamis. Sistem yang memiliki pola kedatangan dinamis memerlukan teknik penjadualan yang lebih rumit [Elsayed dan Boucher, 1994].

II.1. Penjadualan satu mesin [Elsayed dan Boucher, 1994]

Permasalahan penjadualan paling sederhana adalah penjadualan pada satu mesin. Permasalahan ini melibatkan banyak *job* dengan waktu proses yang bervariasi yang harus dikerjakan oleh mesin tunggal. Permasalahannya adalah menentukan *job* yang harus diproses lebih dahulu. Ada beberapa variabel dalam penjadualan satu mesin ini, yaitu :

- Waktu pemrosesan (*processing time*/t_i), yaitu waktu yang dibutuhkan untuk memproses/mengerjakan *job* i
- Waktu penyelesaian (*completion time*/C_i), yaitu waktu yang diperlukan untuk mengerjakan *job* i sampai selesai. Waktu penyelesaian dirumuskan sebagai :

  $C_i = w_i + t_i$

  w_i adalah waktu tunggu *job* i sebelum diproses, yaitu waktu selama mesin masih memproses *job* yang masuk lebih dahulu dan ti adalah waktu pemrosesan *job* i
- Batas waktu (*due date*/d_i), yaitu batas waktu untuk *job* i harus sudah diselesaikan
- *Lateness* (L_i), yaitu waktu keterlambatan *job* i dari *due date* yang telah ditentukan

  $L_i = C_i - d_i$

Lateness menghitung keterlambatan waktu penjadualan dari *due date* yang diberikan. *Lateness* bernilai negatif jika *job* diselesaikan sebelum *due date* dan positif jika telah melebihi *due date*. *Lateness* yang negatif menunjukkan pelayanan yang baik dari sistem dan biasanya tidak berpengaruh pada variabel lain pada sistem, seperti biaya (*cost*). Namun jika *lateness* bernilai positif akan berpengaruh pada biaya yang harus ditanggung oleh sistem. Oleh karena itu aturan yang biasa digunakan adalah hanya keterlambatan yang bernilai positif, yang disebut dengan *tardiness*.

- *Tardiness* (T_i), yaitu waktu keterlambatan dari *job* i jika gagal mencapai *due date* atau nol untuk yang lain

  $T_i = \max [0, L_i]$

II.1.1. Waktu penyelesaian rata – rata (mean flow time)

Permasalahan ini mendasarkan pada kriteria optimasi penjadualan akan dicapai dengan menyusun urutan pemrosesan *job* sedemikian hingga waktu penyelesaian rata – rata untuk setiap *job* dapat diminimalisasi. Waktu penyelesaian rata – rata dapat dirumuskan sebagai berikut :

$$MFT = \frac{\sum_{i=1}^{n} C_i}{n}$$

MFT = waktu penyelesaian rata – rata

Ci = waktu penyelesaian *job* i

n = jumlah *job*

Sebagai contoh adalah jika diberikan lima buah *job* dan satu mesin dengan waktu pengerjaan sebagai berikut : $t_1 = 4$, $t_2 = 3$, $t_3 = 5$, $t_4 = 6$, $t_5 = 9$. Maka dapat dengan jelas bahwa permasalahan tersebut dapat diselesaikan dengan menyusun urutan penyelesaian *job*. Jika urutannya adalah sebagai berikut :

$j_2 - j_4 - j_5 - j_3 - j_1$

maka waktu penyelesaian dapat dilihat pada tabel berikut :

Tabel 2.1
Contoh penyelesaian permasalahan MFT : alternatif 1

job	w_i	t_i	C_i
2	0	3	3
4	3	6	9
5	9	9	18
3	18	5	23
1	23	4	27
	53	27	80

MFT = 80 / 5 =16

Namun jika urutannya adalah : $j_3 - j_4 - j_1 - j_5 - j_2$, maka hasilnya adalah :

Tabel 2.2
Contoh penyelesaian permasalahan MFT : alternatif 2

job	w_i	t_i	C_i
3	0	5	3
4	5	6	11
1	11	4	15
5	15	9	24
2	24	3	27
	55	27	80

MFT = 80 / 5 =16

Jadi, hasilnya adalah sama. Hal ini disebabkan belum ditentukan aturan yang spesifik untuk menyelesaikan permasalahan di atas, sehingga urutan pekerjaan hanya dilakukan random atau mencoba – coba, sehingga dimungkinkan mendapatkan hasil yang tidak optimal. aturan yang lebih spesifik itu misalnya penyelesaian *job* dimulai dari *job* dengan waktu pengerjaan terpendek (*shortest processing time*).

II.1.2. Waktu pengerjaan terpendek (shortest processing time)

Jika diberikan permasalahan seperti di atas, maka penyelesaian penjadualan diberikan urutan *job* sebagai berikut :

$j_2 - j_1 - j_3 - j_4 - j_5$

Dan waktu penyelesaian didapatkan sebagai berikut :

Tabel 2.3
Contoh penyelesaian permasalahan SPT

job	Wi	Ti	Ci
2	0	3	3
1	3	4	7
3	7	5	12
4	12	6	18
5	18	9	27
	40	27	67

MFT = 67 / 5 =13.40

Jadi, dapat dilihat bahwa dengan *shortest processing time* dapat meminimalisasi MFT dan waktu penyelesaian seluruh *job*, sehingga dapat dikatakan jadual yang dihasilkan adalah jadual optimal. Waktu penyelesaian seluruh *job* ini yang disebut *makespan*.

Dengan menerapkan aturan SPT, *lateness* dari *job* juga menjadi minimal. Sebagaimana tersebut di atas *lateness* dari *job* i dirumuskan dengan :

$$L_i = C_i - d_i$$

dan *tardiness* dirumuskan dengan :

$$T_i = \max [0, C_i - d_i]$$

Maka keterlambatan rata – rata (*mean lateness* /L) adalah sebagai berikut :

$$L = \frac{1}{n}\sum_{i=1}^{n} Li = \frac{1}{n}\sum_{i=1}^{n} (C_i - d_i) = \frac{1}{n}\sum_{i=1}^{n} C_i - \frac{1}{n}\sum_{i=1}^{n} d_i$$

$$L = MFT - \overline{d}$$

$\overline{d}$ adalah rata – rata *due date* dari n *job*

Jika kriteria dari penjadualan adalah minimisasi keterlambatan maksimum dari n *job*, maka jadual optimal didapatkan dengan menyusun berdasarkan kenaikan *due date* :

$$d_1 \leq d_2 \leq d_3 \leq d_4 \leq \ldots \leq d_n$$

II.2. Penjadualan flow shop

Penjadualan *flow shop* memiliki karakteristik sebagai berikut :

- Ada n *job* dan m mesin
- Setiap *job* harus diproses pada semua mesin dengan urutan yang sama pengerjaannya untuk setiap mesin
- Setiap *job* hanya diproses pada satu mesin pada waktu yang sama
- Setiap mesin hanya mengerjakan satu *job* pada waktu yang sama

Tujuan penjadualan *flow shop* adalah menentukan *makespan* dengan menggunakan ketentuan di atas.

Jika diberikan permasalahan penjadualan *flow shop* dengan 6 *job* dan 2 mesin dengan ketentuan pada tabel berikut :

Tabel 2.4
Contoh permasalahan *flow shop* 6x2

Mesin	Job					
	1	2	3	4	5	6
1	2	2	3	1	4	2
2	4	1	3	2	4	1

Dan jika $C(j_i,k)$ menyatakan waktu penyelesaian job j_i pada mesin k, dan $\{j_1, j_2,\ldots, j_n\}$ adalah urutan pekerjaan, maka makespan dapat dihitung dengan perhitungan sebagai berikut :

$C(j_1, 1) = t_{j1}1$

$C(j_1, k) = C(j_1, k\text{-}1) + t_{j1}k \qquad k = 2, 3,\ldots, m$

$C(j_i, 1) = C(j_{i\text{-}1}, 1) + t_{ji}1 \qquad i = 2, 3,\ldots, n$

$C(j_i, k) = \max(C(j_{i\text{-}1}, k), C(j_i, k\text{-}1)) + t_{ji}k \qquad i = 2,3,\ldots,n \qquad k = 2, 3,\ldots,m$

Jadi, *makespan* dapat ditentukan sebagai : $C_{max} = C(j_n, m)$.

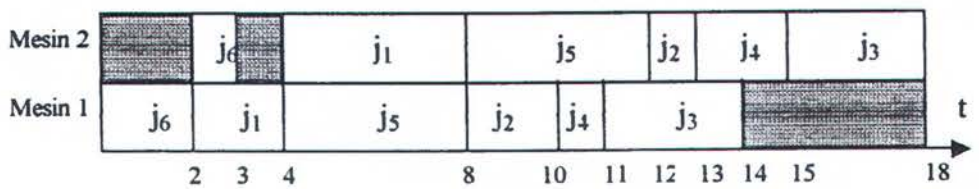
Maka bila penjadualan dilakukan dengan urutan pengerjaan *job* : 6 – 1 – 5 – 2 – 4 – 3, maka *makespan* dapat dihitung sebagai berikut :

Tabel 2.5
Contoh penghitungan *makespan*

Job j_i	6	1	5	2	4	3
C(ji,1)	2	4	8	10	11	14
C(ji,2)	3	8	12	13	15	18

Jadi, *makespan* = 18

Dan diagram *Gant-chart* untuk permasalahan tersebut adalah :



Gambar 2.1 Contoh penjadualan *flow shop* 6x2

II.2.1. Aturan Johnson [Mortton dan Pentico, 1993]

Penjadualan *flow shop* dengan jumlah mesin = 2 diselesaikan dengan metode yang dikenal dengan aturan Johnson. Jika terdapat *job* j dan waktu pengerjaan adalah t_j1 pada mesin 1 dan t_j2 pada mesin 2, maka aturan Johnson didapatkan sebagai berikut :

- Bentuklah himpunan U = $\{j \mid t_j1 < t_j2\}$ dan himpunan V = $\{j \mid t_j1 \geq t_j2\}$
- Urutkan *job* dalam himpunan U mulai dari t_j1 terkecil

- Urutkan *job* dalam himpunan V mulai dari t_j2 terbesar
- Jadual didapat dari union himpunan U dan himpunan V

Bila permasalahan *flow shop* di atas diselesaikan dengan aturan Johnson, maka jadual yang didapat adalah :

- Himpunan U = {1,4} dan v = {2, 3, 5, 6}
- Setelah diurutkan didapat U = {4, 1} dan V = {5, 3, 2, 6}
- Union U dan V adalah {4, 1, 5, 3, 2, 6}
- Jadi, didapat jadual pengerjaan *job* : 4 – 1 – 5 – 3 – 2 – 6

II.2.2. Metode Palmer [Cheng dan Gen, 1996]

Palmer memperkenalkan *slope index* untuk mengurutkan *job* berdasarkan waktu pengerjaannya. Dalam mengurutkan *job* diprioritaskan *job* dengan waktu pengerjaan yang cenderung meningkat dari mesin satu ke mesin lain. *Slope index* dapat dihitung sebagai berikut :

$$S_i = \sum_{j=1}^{m} (2j - m - 1)t_{ij}, \qquad i = 1, 2, \ldots, n$$

Jadual optimal diperoleh dengan mengurutkan *job* berdasarkan S_i sedemikian sehingga $S_{i1} \geq S_{i2} \geq S_{i3} \geq \ldots \geq S_{in}$

II.2.3. Metode Nawaz-Escore-Ham [Cheng dan Gen, 1996]

Metode ini diperkenalkan oleh Nawaz, Escore, dan Ham. Metode ini mendasarkan pada asumsi bahwa *job* dengan waktu total pengerjaan lebih tinggi mempunyai prioritas lebih tinggi untuk dijadualkan. Waktu total adalah waktu pengerjaan pada seluruh mesin.

Berikut ini adalah algoritma Nawaz-Escore-Ham (NEH) :

- Urutkan mulai waktu pengerjaan total tertinggi sampai terendah
- Ambil dua *job* pertama dan atur keduanya sehingga *makespan* minimum
- Untuk k=3 sampai selesai, masukkan *job* k ke dalam jadual pada tempat yang sesuai sedemikian hingga *makespan* minimum.

II.3. Penjadualan job shop

Penjadualan *job shop* seringkali dirumuskan sebagai berikut : Ada beberapa pekerjaan (*job*) dan beberapa mesin. Setiap *job* tersusun atas beberapa operasi yang merupakan rangkaian operasi. Setiap operasi tersebut harus dikerjakan pada mesin yang disediakan selama selang waktu tertentu dan pada operasi tidak dapat disisipkan masa istirahat selama pengerjaan *job*. Sedangkan setiap mesin hanya dapat mengerjakan sebuah operasi setiap waktu. Sistem penjadualan berfungsi membagi operasi – operasi yang ada kepada mesin – mesin yang disediakan untuk dikerjakan selama selang waktu tertentu, sehingga ditemukan waktu yang minimum untuk mengerjakan seluruh *job* yang disebut *makespan* [Cheng dan Gen, 1996].

II.3.1. Penjadualan job shop klasik

Penjadualan *job shop* klasik memiliki batasan sebagai berikut :

- Permasalahan terdiri atas n *job* dan m mesin
- Setiap *job* tidak diproses dua kali oleh mesin yang sama
- Tidak ada operasi yang harus diutamakan/pendahulu antara *job* yang berbeda. Ini menunjukkan bahwa antara *job* mempunyai sifat *independen*, tetapi pada *job* yang sama untuk setiap operasi harus tetap urut pengerjaannya atau harus memperhatikan operasi pendahulunya.
- Pada operasi tidak dapat disisipkan masa istirahat dan harus diproses sampai selesai
- Setiap mesin hanya dapat memproses sebuah *job* pada waktu yang sama
- Setiap *job* hanya diproses oleh sebuah mesin pada waktu yang sama
- Tidak ada batas waktu penundaan pengerjaan seluruh *job* karena diasumsikan tidak ada penundaan/keterlambatan

Tujuan penjadualan *job shop* adalah menentukan makespan. Perbedaan penjadualan ini dengan penjadualan *flow shop* adalah tidak adanya ketentuan bahwa urutan pengerjaan *job* pada setiap mesin harus sama.

II.3.2. Rumusan penjadualan job shop

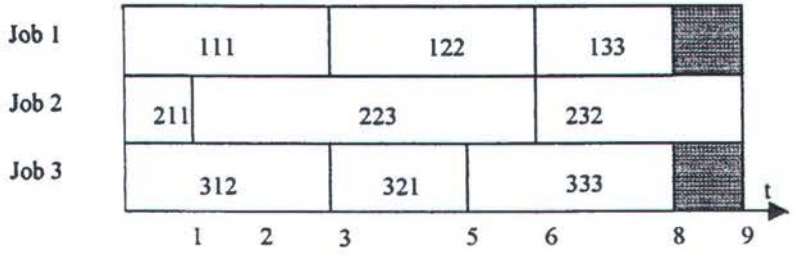
Penjadualan *job shop* dapat diilustrasikan dengan menggunakan diagram *Gant-chart*. Untuk menggambarkan permasalahan digunakan dua buah *Gant-chart*, yaitu digram pertama untuk menunjukkan pemrosesan pekerjaan tiap satuan waktu dan diagram kedua menunjukkan aktivitas mesin tiap satuan waktu.

Misalkan diberikan tiga pekerjaan (*job*) dan disediakan tiga mesin untuk memprosesnya selama beberapa satuan waktu tertentu. Agar dapat digambarkan pada *Gant-chart*, maka digunakan indeks jom dengan ketentuan j menunjukkan *job*, o menunjukkan operasi dari *job* j, dan m menunjukkan mesin yang memproses operasi tersebut. Berikut ini tabel permasalahan di atas :

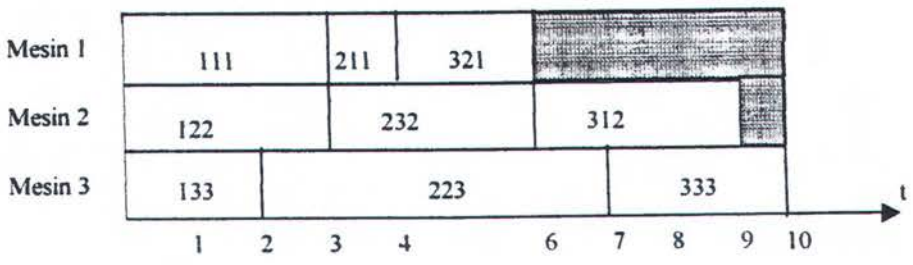
Tabel 2.6
Contoh permasalahan *job* shop 3x3

	Waktu proses				Urutan mesin		
	Operasi				Operasi		
Job	1	2	3	Job	1	2	3
1	3	3	2	1	1	2	3
2	1	5	3	2	1	3	2
3	3	2	3	3	2	1	3

Jika digambarkan diagramnya dalam *Gant-chart* adalah sebagai berikut :



Gambar 2.2 Gant-chart pemrosesan *job*

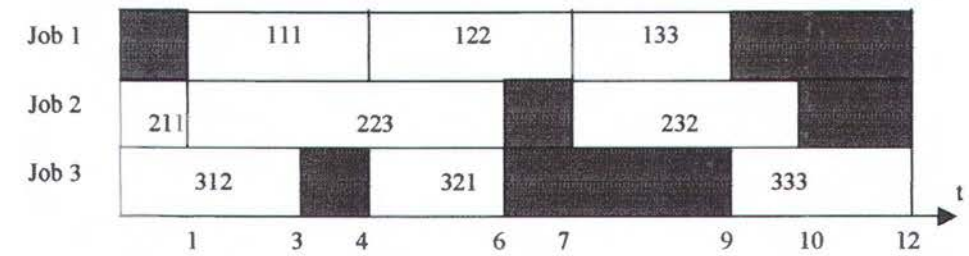


Gambar 2.3 Gant-chart aktivitas mesin

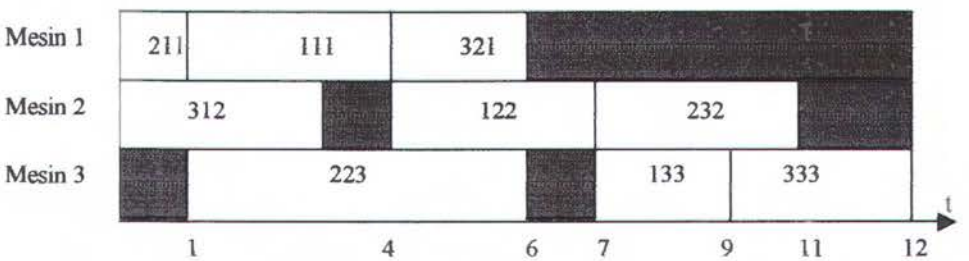
Namun pada diagram di atas belum dapat dikatakan sebagai jadual yang layak (*infeasible*) karena beberapa operasi dalam *job* yang dikerjakan oleh satu mesin secara simultan, yaitu operasi pada suatu *job* sudah dimulai dikerjakan pada saat operasi yang lain pada *job* itu belum selesai dikerjakan pada mesin yang sama. Hal ini tidak memenuhi ketentuan bahwa setiap mesin hanya memproses sebuah *job* pada saat yang sama. Contohnya pada gambar II.2. adalah *job* dengan indeks 111 dan 211 menunjukkan bahwa mesin 1 menerima *job* 2 pada saat mesin 1 belum selesai mengerjakan *job* yang lain (*job* 1).

Demikian juga bila dilihat pada gambar II.3. yang menggambarkan jadual dari sisi aktivitas pengerjaan mesin. Jadual ini juga dikatakan *infeasible*. Pada *job* dengan indeks 111 dan 132 menunjukkan bahwa *job* 1 dikerjakan dalam waktu bersamaan oleh mesin 1 dan mesin 2. *Job* 1 dikerjakan secara simultan oleh dua buah mesin. Hal ini tidak memenuhi ketentuan bahwa setiap *job* hanya diproses oleh sebuah mesin pada saat yang sama.

Sehingga diperlukan penjadualan yang dapat memenuhi ketentuan tersebut di atas [Mortton dan Pentico, 1993]. Salah satu jadual yang *feasible* yang dapat dibuat adalah sebagai berikut :



Gambar 2.4 Gant-chart (*feasible*) pemrosesan *job*



Gambar 2.5 Gant-chart (*feasible*) aktivitas mesin

II.3.2.1. Jenis jadual

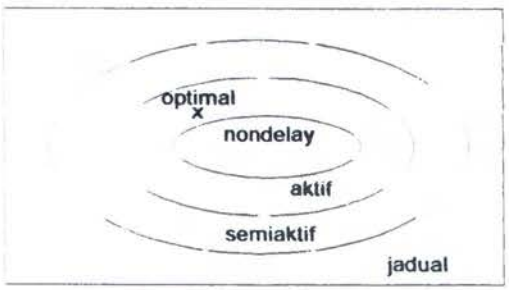
Penyusunan jadual yang *feasible* mengakibatkan adanya mesin yang tidak memproses *job* pada suatu waktu tertentu, yang disebut *idle*. Pada dasarnya terdapat banyak jadual yang *feasible* karena sebarang waktu *idle* tersebut dapat disisipkan di antara operasi – operasi pada sebarang mesin. Oleh karena itu dapat tercipta beberapa jenis jadual. Dan operasi – operasi yang ada pada jadual, pada

Gant-chart, dapat digeser ke kiri/lebih awal sejauh mungkin tanpa mengubah urutan operasi pada setiap *job* karena dibatasi oleh operasi pendahulu. Cara ini disebut *limited left-shift* atau *local left-shift* [Baker, 1974]. Jika sebuah *shift* menyebabkan berubahnya urutan operasi pada setiap mesin disebut *global left-shift*. Dengan *global left-shift*, operasi diizinkan melompati urutan operasi sebelumnya selama interval waktu masih memungkinkan untuk ditempati. Yang dimaksud melompati operasi sebelumnya adalah operasi yang bersangkutan digeser sedemikian hingga dikerjakan lebih awal karena adanya waktu *idle* pada mesin tertentu. Hal ini diizinkan karena untuk operasi pada *job* yang berbeda sifatnya *independen*.

Berdasarkan konsep *left-shift* di atas, maka jadual dapat dibedakan menjadi :

- Jadual semiaktif, jika tidak dapat ditemukan lagi operasi yang dapat digeser dengan *local left-shift*. Jadual ini jumlah kemungkinannya masih terlalu besar
- Jadual aktif, jika tidak dapat ditemukan lagi operasi yang dapat digeser dengan *global left-shift*.
- Jadual nondelay, jika tidak ada mesin yang dibiarkan *idle* pada saat mengerjakan *job*. Jadual nondelay memang lebih kecil dari jadual aktif, tetapi tidak menjamin akan terdapat jadual yang optimal. Jadual optimal akan didapatkan pada jadual aktif [Baker, 1974].

Berikut adalah diagram Venn yang menunjukkan hubungan jenis – jenis jadual tersebut di atas :



Gambar 2.6 Diagram Venn jenis jadual

II.3.2.2. Pemrograman integer

Dua batasan yang harus diperhatikan untuk penjadualan *job shop* klasik adalah :

- Operasi yang harus diutamakan pada *job* yang diberikan

  Hal ini berkaitan dengan batasan bahwa setiap *job* hanya dikerjakan pada satu mesin pada waktu yang sama. Jadi, harus diperhatikan pada waktu tertentu apakah ada mesin lain yang sedang mengerjakan operasi pada *job* yang sama
- Operasi yang tidak menyebabkan mesin bekerja secara *overlapping*

  Hal ini berkaitan dengan batasan bahwa setiap mesin hanya mengerjakan satu *job* pada waktu yang sama. Jadi, harus diperhatikan pada waktu tertentu apakah ada mesin lain yang sedang mengerjakan *job* yang sama.

Jika c_{jk} adalah waktu penyelesaian *job* j pada mesin k dan t_{jk} adalah waktu pemrosesan *job* j pada mesin k. Maka bila terdapat *job* i yang harus diproses pada

mesin h dan mesin k dengan syarat pada mesin h yang diutamakan, didapatkan batasan : $c_{ik} - t_{ik} \geq c_{ih}$

Jika pada mesin k yang harus diutamakan, maka batasan yang didapat adalah :

$$c_{ih} - t_{ih} \geq c_{ik}$$

Jika a_{ihk} adalah koefisien indikator, maka

$$a_{ihk} \begin{cases} 1, \text{ jika proses pada mesin h yang diutamakan} \\ 0, \text{ jika proses pada mesin k yang diutamakan} \end{cases}$$

Jadi, batasan di atas dapat dituliskan kembali sebagai :

$$c_{ik} - t_{ik} + M(1 - a_{ihk}) \geq c_{ih}, \qquad i = 1, 2, 3, \ldots, n$$

$$h, k = 1, 2, 3, \ldots, m$$

M adalah bilangan positif yang sangat besar

Jika *job* i dan *job* j dikerjakan pada mesin k, maka bila *job* i datang sebelum *job* j, didapatkan batasan : $c_{jk} - c_{ik} \geq t_{jk}$

Jika *job* j yang mendahului, didapatkan batasan : $c_{ik} - c_{jk} \geq t_{jk}$

Dan jika x_{ijk} adalah koefisien indikator, maka :

$$x_{ijk} \begin{cases} 1, \text{ jika } \textit{job} \text{ i yang mendahului} \\ 0, \text{ jika } \textit{job} \text{ j yang mendahului} \end{cases}$$

Jadi, batasan di atas dapat dituliskan kembali sebagai :

$$c_{jk} - c_{ik} + M(1 - x_{ijk}) \geq t_{jk}, \qquad i, j = 1, 2, 3, \ldots, n$$

$$k = 1, 2, 3, \ldots, m$$

Sehingga penjadualan *job shop* dengan fungsi tujuan dapat dituliskan sebagai berikut :

$$\min_{1 \le k \le m} \max_{1 \le k \le m} \left\{ \max \{cik\} \right\}$$

batasan : $c_{ik} - t_{ik} + M(1 - a_{ihk}) \geq c_{ih}$, $i = 1, 2, 3, \ldots, n$

$h, k = 1, 2, 3, \ldots, m$

$c_{jk} - c_{ik} + M(1 - x_{ijk}) \geq t_{jk}$, $i, j = 1, 2, 3, \ldots, n$

$k = 1, 2, 3, \ldots, m$

$c_{ik} \geq 0$, $i = 1, 2, 3, \ldots, n$ $k = 1, 2, 3, \ldots, m$

$x_{ijk} = 0$ atau 1, $i, j = 1, 2, 3, \ldots, n$ $k = 1, 2, 3, \ldots, m$

II.3.2.3. Pemrograman linier

Jika N = {0, 1, 2, 3, ..., n} adalah himpunan operasi yang terdapat dalam *job* dengan 0 sebagai "start" dan n sebagai "finish" dan M = {0, 1, 2, 3, ..., m} adalah himpunan mesin; sedangkan A menunjukkan himpunan (i, j) dengan i adalah operasi pada suatu *job* yang hanya dapat dikerjakan setelah operasi j dan E_k adalah himpunan (i, j) dengan i adalah operasi yang dijadualkan pada mesin k dengan syarat tidak bersamaan dengan waktu operasi j, maka untuk setiap operasi i dengan waktu pemrosesan = d_i dan waktu dimulainya operasi = t_i, dapat dirumuskan penjadualan *job shop* adalah :

$\min t_n$ ..... (i)

batasan : $t_j - t_i \geq d_i$, $(i, j) \in A$ ..... (ii)

$t_j - t_i \geq d_i$ atau $t_i - t_j \geq d_i$, $(i, j) \in E_k$, $k \in M$ ..... (iii)

$t_j \geq 0$ ..... (iv)

(i) adalah fungsi tujuan, yaitu minimum *makespan*

(ii) menunjukkan urutan pengerjaan operasi pada suatu *job*

(iii) menunjukkan setiap mesin hanya mengerjakan sebuah *job* pada waktu yang sama

II.3.3. Pendekatan heuristik

Prosedur penjadualan heuristik untuk penjadualan *job shop* adalah dengan menerapkan aturan prioritas. Aturan prioritas ini digunakan pada saat menempatkan operasi tertentu ke dalam jadual. Aturan prioritas itu adalah :

- SPT (*short processing time*), yaitu pemilihan operasi dengan waktu proses terpendek
- LPT (*longest processing time*), yaitu pemilihan operasi dengan waktu proses terlama
- FCFS (*frst come first served*), yaitu pemilihan operasi yang lebih awal pada urutan *job* pada satu mesin
- MWR (*most work remaining*), yaitu pemilihan operasi pada *job* yang memiliki sisa total waktu pengerjaan terbanyak

- MOR (*most operation remaining*), yaitu pemilihan operasi pada *job* yang memiliki sisa jumlah operasi terbanyak
- LWR (*least work remaining*), yaitu pemilihan operasi pada *job* yang memiliki sisa total waktu pengerjaan paling sedikit
- LOR (*least operation remaining*), yaitu pemilihan operasi pada *job* yang memiliki sisa jumlah operasi paling sedikit
- Random, yaitu pemilihan operasi secara random

Untuk dapat menyusun prosedur heuristik penjadualan *job shop* tersebut, sebelumnya dijelaskan variabel yang terlibat di dalamnya. Variabel tersebut adalah :

- PS_t = jadual parsial yang mencakup t operasi yang telah terjadual
- S_t = operasi – operasi yang bisa dijadualkan pada tahap t, yang berhubungan dengan PS_t sebelumnya
- τ = waktu paling awal di mana operasi $j \in S_t$ bisa dimulai
- ϕ_j = waktu paling awal di mana operasi $j \in S_t$ bisa diselesaikan

Dan prosedur penjadualan untuk jadual aktif adalah sebagai berikut :

Langkah 1 Beri harga $t = 0$ dan mulai dengan PS_t sebagai jadual parsial *null*. Inisialisasi S_t adalah seluruh operasi tanpa operasi pendahuluan

Langkah 2 Tentukan $\phi_t^* = \min_{j \in S_t} \{\phi_j\}$ dan mesin m* untuk melaksanakan ϕ_t^*

Langkah 3 Untuk setiap operasi $j \in S_t$ yang memerlukan mesin m* dan $\tau < \phi_t^*$, hitung indeks prioritas menurut aturan prioritas tertentu.

Dapatkan operasi dengan indeks terkecil dan tambahkan pada PS_t seawal mungkin, sehingga terbentuk sebuah jadual parsial PS_{t+1}

Langkah 4 Untuk membentuk jadual parsial PS_{t+1}, data diubah dengan ketentuan sebagai berikut :

1. Hapus operasi - operasi j dari S_t
2. Bentuk S_{t+1} dengan menambah operasi pengikut dari operasi j pada S_t
3. $t = t + 1$

Langkah 5 Mengulangi mulai langkah 2 sampai seluruh jadual terbentuk

Jika dibentuk jadual nondelay, maka prosedur penjadualan sebagai berikut :

Langkah 1 Beri harga $t = 0$ dan mulai dengan PS_t sebagai jadual parsial *null*. Inisialisasi S_t adalah seluruh operasi tanpa operasi pendahuluan

Langkah 2 Tentukan $\tau_t^* = \min_{j \in S_t}\{\tau_j\}$ dan mesin m^* untuk melaksanakan ϕ_t^*

Langkah 3 Untuk setiap operasi $j \in S_t$ yang memerlukan mesin m^* dan $\tau = \tau_t^*$, hitung indeks prioritas menurut aturan prioritas tertentu. Dapatkan operasi dengan indeks terkecil dan tambahkan pada PS_t seawal mungkin, sehingga terbentuk sebuah jadual parsial PS_{t+1}

Langkah 4 Untuk membentuk jadual parsial PS_{t+1}, data diubah dengan ketentuan sebagai berikut :

1. Hapus operasi - operasi j dari S_t
2. Bentuk S_{t+1} dengan menambah operasi pengikut dari operasi j pada S_t
3. $t = t + 1$

Langkah 5 Mengulangi mulai langkah 2 sampai seluruh jadual terbentuk

BAB III

ALGORITMA GENETIK

Pada penjelasan di bawah ini akan diuraikan beberapa hal mengenai konsep dasar dan cara kerja algoritma genetik.

III.1. Konsep dasar algoritma genetik

Pada penjelasan tentang konsep dasar algoritma genetik berikut ini akan diperkenalkan beberapa istilah yang digunakan dalam algoritma genetik sebagai terminologi untuk memahami kinerja algoritma genetik, penjelasan secara garis besar tentang algoritma genetik dalam bentuk struktur umum algoritma genetik, dan karakteristik algoritma genetik dalam hal memanipulasi solusi, serta kemiripannya dengan prinsip algoritma pencarian (*searching*) konvensional.

III.1.1. Terminologi

Pada algoritma genetik yang menggunakan konsep dasar meniru proses evolusi makhluk hidup, maka dalam terminologi yang digunakan juga banyak menggunakan istilah yang diadaptasi dari ilmu biologi, khususnya genetika.

Menurut teorema yang lazim digunakan dalam ilmu genetika, bahwa untuk sebuah organisme tersusun atas komponen yang menyebabkan perbadaan khas antara organisme yang satu dengan yang lain. Komponen itu disebut kromosom. Satu atau lebih kromosom menyusun sebuah organisme yang khas/spesifik.

Kromosom yang lengkap beserta unsur – unsurnya disebut juga genotip. Sedangkan perwujudan organismenya disebut fenotip.

Pada algoritma genetik, istilah populasi diberikan untuk menamai himpunan solusi. Untuk pertama kalinya, himpunan solusi dihasilkan secara acak (*random*). Di dalam populasi terdapat individu dan setiap individu tersebut merupakan representasi dari sebuah solusi. Setiap individu disebut juga kromosom. Setiap kromosom direpresentasikan dengan string atau rangkaian simbol. Dalam banyak permasalahan representasi tersebut dalam bentuk *bit string*. Setiap kromosom tersusun atas bagian – bagian yang terstruktur yang disebut gen. Setiap gen menempati lokasi yang dinamakan *locus* dan memiliki nilai (*value*) yang disebut *allele*. Setiap nilai gen inilah yang merupakan pengkodean variabel permasalahan yang akan diselesaikan.

III.1.2. Struktur umum

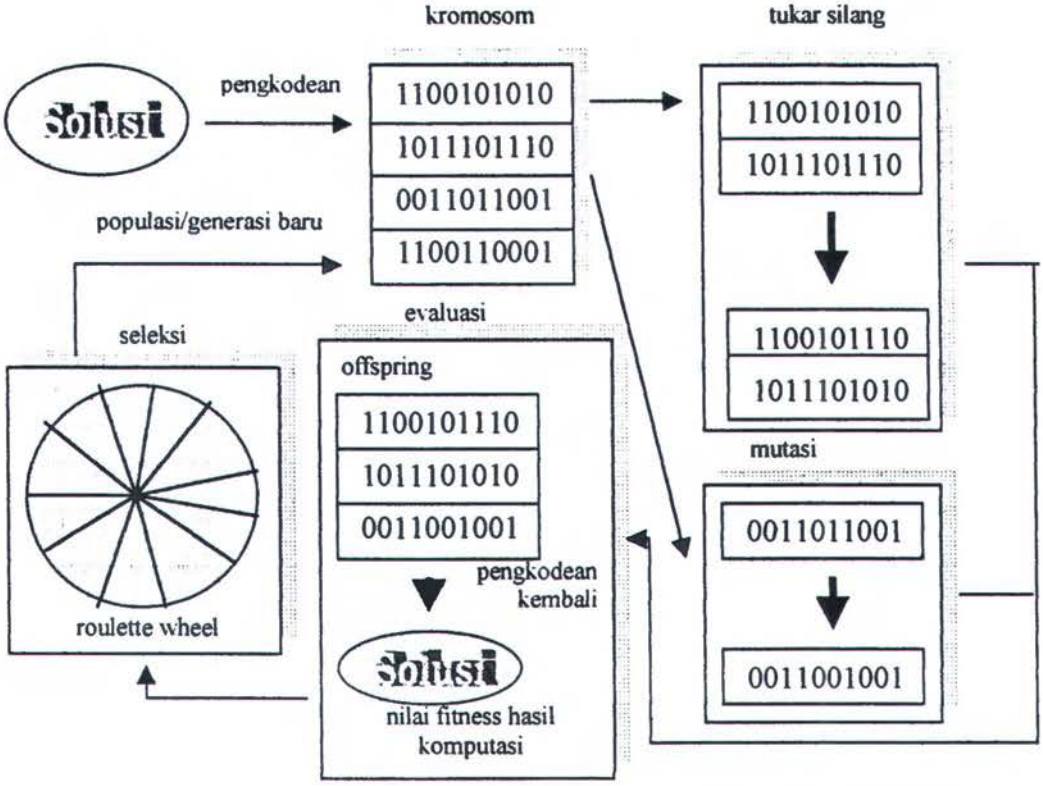
Algoritma genetik berawal dari inisialisasi himpunan solusi yang didapat secara acak. Himpunan solusi awal ini disebut dengan istilah populasi awal. Kromosom – kromosom yang terdapat pada populasi awal akan berevolusi menurut proses iterasi yang berkelanjutan yang disebut generasi. Generasi dapat diartikan sebagai populasi baru hasil iterasi. Pada setiap generasi, kromosom – kromosom dievaluasi berdasarkan suatu takaran tertentu yang disebut *fitness*.

Sedangkan pada proses iterasinya, diberlakukan dua jenis operasi untuk menghasilkan generasi – generasi berikutnya. Kedua jenis operasi itu adalah :

- Operasi genetik (*genetic operations*), yang terdiri atas tukar silang (*crossover*) dan mutasi (*mutation*), dan
- Operasi evolusi (*evolution operation*), yaitu seleksi (*selection*).

Pembentukan generasi berikutnya diawali dengan membentuk kromosom – kromosom baru yang disebut *offspring*. *Offspring* dibentuk dengan melalui dua operasi genetik, yaitu tukar silang dan mutasi. Setelah melalui kedua operasi tersebut, *offspring* yang dihasilkan bersama dengan beberapa kromosom lama (*parents*) memasuki tahap evaluasi dan dilanjutkan dengan seleksi. Evaluasi adalah menetapkan nilai kromosom – kromosom yang disebut nilai *fitness* untuk kemudian dilakukan seleksi. Seleksi adalah pemilihan *offspring* dan/atau *parents* berdasarkan suatu nilai *fitness*-nya. Kromosom – kromosom yang memiliki *fitness value* lebih tinggi mempunyai kemungkinan lebih tinggi untuk dipilih sebagai anggota populasi baru. Populasi baru ini yang digunakan pada proses regenerasi berikutnya. Setelah melalui beberapa generasi, algoritma akan konvergen pada kromosom terbaik pada generasi tertentu [Cheng dan Gen, 1996]. Kromosom terbaik inilah yang diharapkan merupakan solusi untuk permasalahan yang dihadapi.

Untuk lebih jelasnya, diberikan ilustrasi sebagai berikut :



Gambar 3.1 Struktur umum algoritma genetik

Jika diketahui P(t) adalah *parents* dan C(t) adalah *offspring* pada suatu generasi t, maka struktur umum algoritma genetik dapat dituangkan ke dalam suatu prosedur sebagai berikut :

procedure genetic algorithms;
begin
t = 0;
inisialisasi P(t);
evaluasi P(t);

```
while (bukan kondisi terminasi) do
begin
    rekombinasi P(t) menjadi C(t); {operasi tukar silang dan mutasi}
    evaluasi C(t);
    dapatkan P(t+1); {hasil seleksi P(t) dan seleksi C(t)};
    t = t +1;
end;
end;
```

III.1.3. Eksploitasi dan eksplorasi

Teknik pemecahan masalah yang secara universal dan sering digunakan adalah pencarian (*searching*). Terdapat beragam macam teknik pencarian, tetapi secara umum dapat dibedakan menjadi dua macam strategi berdasarkan karakteristiknya. Strategi pertama disebut *blind search*, yaitu strategi pencarian yang tidak memerlukan informasi tambahan tentang domain problem yang dihadapi. Sedangkan strategi kedua dikenal dengan *heuristic search*, yaitu strategi pencarian yang memerlukan tambahan informasi untuk mengarahkan proses pencariannya ke dalam daerah pencarian terbaik, sehingga lebih terarah untuk menemukan solusi terbaik dan optimal.

Pada teknik pencarian yang telah dikenal dan beragam tersebut, pada dasarnya memiliki dua karakteristik utama di dalam proses pencarian atau iterasi yang dilakukan, yaitu mengeksploitasi solusi terbaik dan mengeksplorasi daerah pencarian [Booker, 1987].

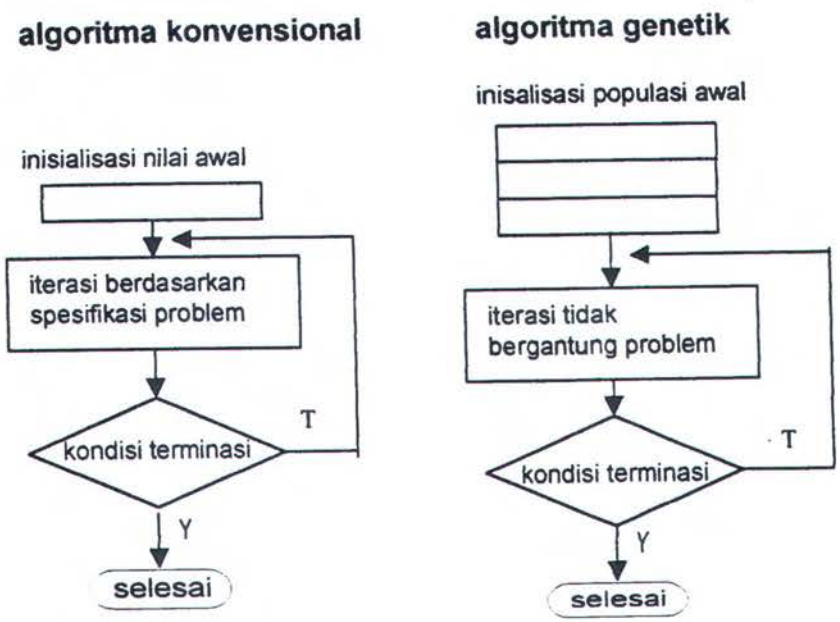
Teknik *hill climbing* adalah salah satu contoh teknik pencarian yang mengeksploitasi solusi terbaik dengan terus – menerus melakukan iterasi sejauh

iterasi itu masih mungkin dilakukan. Namun teknik ini mengabaikan eksplorasi daerah pencarian. Sedangkan contoh untuk teknik pencarian yang mengeksplorasi daerah pencarian adalah *random search*. Namun teknik ini mengabaikan eksploitasi solusi terbaik dalam iterasinya. Algoritma genetik merupakan metode yang menggabungkan keduanya dengan menyeimbangkan eksplorasi daerah pencarian dan eksploitasi solusi terbaik [Michalewicz, 1996].

Berawal dari daerah solusi yang sangat luas yang dihasilkan secara random dan disebut populasi. Kemudian melalui proses tukar silang, seluruh daerah solusi dieksplorasi. Selain itu proses tukar silang juga melakukan iterasi yang mengeksploitasi solusi terbaik dengan mempertahankan solusi yang memiliki nilai *fitness* yang lebih tinggi untuk dipilih sebagai *parents* pada iterasi berikutnya.

III.1.4. Pencarian berbasis populasi

Pada umumnya algoritma pemecahan masalah optimasi merupakan urutan langkah menuju solusi optimal. Berawal dari satu nilai yang kemudian dieksploitasi menurut iterasi tertentu. Sedangkan algoritma genetik berawal dari beberapa nilai yang disebut populasi awal. Kemudian melalui proses iterasi tertentu bergerak dari populasi yang satu kepada populasi yang diturunkan berikutnya. Untuk lebih jelasnya dapat dilihat pada diagram alir di bawah ini :



Gambar 3.2. Diagram alir perbandingan algoritma konvensional dengan algoritma genetik

III.2. Cara kerja algoritma genetik

Yang dimaksud dengan cara kerja algoritma genetik pada pembahasan berikut ini adalah urut – urutan proses penyelesaian masalah dengan menggunakan algoritma genetik seperti yang telah dijelaskan menurut struktur umum algoritma genetik. Proses diawali dengan representasi masalah, iterasi dengan operasi genetik dan operasi evolusi, serta fungsi evaluasi untuk menilai kromosom yang masih dapat dipertahankan pada setiap generasinya.

III.2.1. Representasi

Hal yang paling mendasar pada pemecahan masalah dengan menggunakan algoritma genetik adalah merepresentasikan atau mengkodekan masalah ke dalam bentuk kromosom – kromosom solusi. Sebuah kromosom merupakan representasi dari sebuah solusi. Kromosom merupakan rangkaian gen yang memuat nilai terkodekan, sehingga kromosom dapat disebut juga dengan *string*. Setiap gen memuat nilai tertentu yang terkodekan menurut cara representasi tertentu. Pada umumnya representasi yang sering digunakan oleh para peneliti adalah dengan *bit string*.

III.2.1.1. Notasi skema

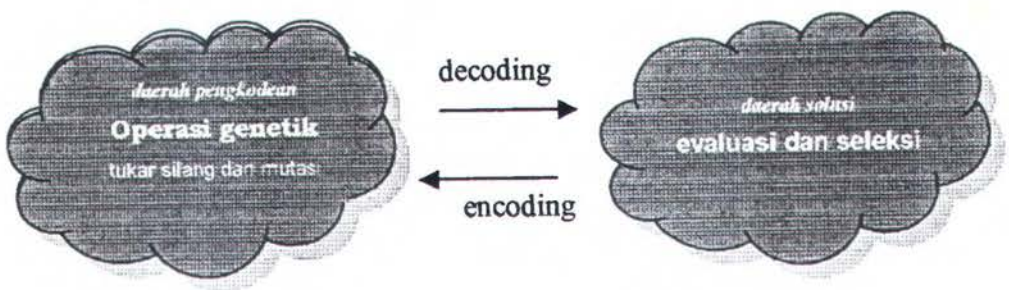
Dasar teori yang dapat digunakan untuk menjelaskan cara representasi solusi dengan *bit string* dikenal dengan notasi skema [Holland, 1975]. Di dalam notasi skema ini digunakan asumsi bahwa setiap *bit string* dapat dibentuk dengan rangkaian tiga digit/simbol, yaitu '0', '1', dan '*'. Simbol '*' mempunyai makna *don't care*.

Misalnya terdapat notasi skema (*111100100). Menurut notasi tersebut akan terdapat kemungkinan dibentuk dua *bit string*, yaitu (0111100100) dan (1111100100). Dan bila terdapat notasi skema (*1*1100100), akan dapat dibentuk empat buah *bit string*, yaitu (0101100100), (01111100100), (1101100100), dan (1111100100). Sehingga dapat disimpulkan bahwa bila terdapat notasi skema dengan n simbol * akan dapat dibentuk 2^n *bit string*. Dan sebaliknya bila terdapat *bit string* dengan panjang m akan dapat dinotasikan sebanyak 2^m notasi skema.

Jika skema dinyatakan dengan S dan order skema (o(S)) adalah jumlah '0' dan '1' yang terdapat pada skema S, maka bila terdapat skema S_1=(***000*110) dan S_2=(****000*0*), dapat dinyatakan $o(S_1)$=6 dan $o(S_2)$=4. Dan jika panjang skema S ($\delta(S)$) adalah jarak antara *fixed bit* ('0' atau '1') yang pertama dan terakhir, maka $\delta(S_1)$=6 dan $\delta(S_2)$=4. Sedangkan jumlah *string* dalam suatu populasi yang cocok dengan skema S pada waktu t dinyatakan dengan $\varepsilon(S,t)$ [Michalewicz, 1996].

III.2.1.2. Daerah pengkodean dan daerah solusi

Algoritma genetik bekerja pada daerah pengkodean dan daerah solusi. Operasi genetik yang terdiri atas tukar silang dan mutasi bekerja pada daerah pengkodean. Sedangkan fungsi evaluasi dan seleksi bekerja pada daerah solusi [Cheng dan Gen, 1996].



Gambar 3.3 Daerah solusi dan daerah pengkodean

Dalam hubungannya dengan kedua daerah tersebut, terdapat beberapa fenomena yang harus diperhatikan, yaitu :

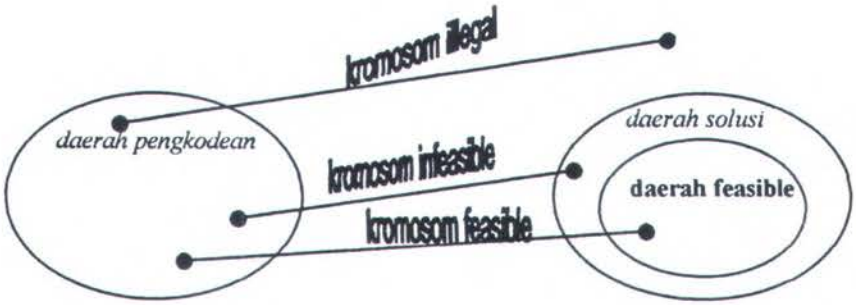
1. Feasibilitas kromosom
2. Legalitas kromosom
3. Pemetaan yang unik

Feasibilitas kromosom untuk menunjukkan apakah sebuah solusi yang dikodekan sudah tepat berada dalam daerah yang *feasible* di antara *domain* permasalahan. Sedangkan legalitas kromosom menunjukkan pada kenyataan bahwa sebuah kromosom benar – benar dalam merepresentasikan sebuah solusi di dalam *domain*. Kemudian dengan merujuk bahwa setiap solusi selalu dikodekan pada setiap kromosom, maka antara solusi dan kromosom terdapat suatu hubungan pemetaan.

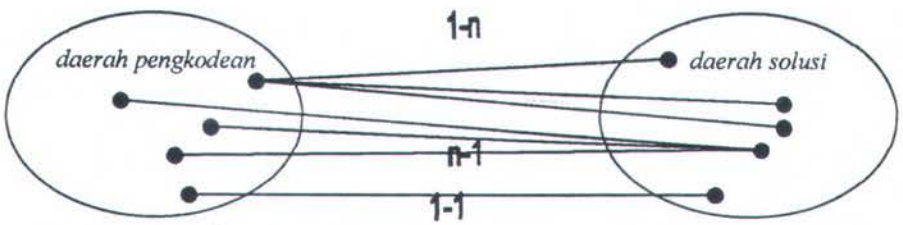
Ada tiga jenis pemetaan yang mungkin antara kromosom dan solusi, yaitu :

1. Pemetaan 1-1
2. Pemetaan n-1
3. Pemataan 1-n.

Pemetaan antara solusi dan kromosom yang terbaik adalah bila di antara keduanya memiliki relasi pemetaan 1-1.



Gambar 3.4 Feasibilitas dan legalitas



Gambar 3.5 Pemetaan kromosom-solusi

III.2.2. Fungsi evaluasi

Yang dimaksud dengan fungsi evaluasi di sini adalah suatu fungsi yang memberikan penilaian berupa nilai *fitness* (*fitness value*) kepada kromosom. Bila pada awal iterasi nilai *fitness* semua kromosom ini mempunyai rentang yang lebar, seiring dengan bertambahnya generasi, beberapa kromosom mempunyai

rentang nilai *fitness* semakin kecil dan mendominasi populasi. Hal ini menyebabkan fungsi evaluasi menjadi masalah yang krusial dalam algoritma genetik karena dominasi kromosom dalam populasi ini dapat menyebabkan iterasi hanya terfokus pada kromosom – kromosom tersebut sebagai akibat tingginya nilai *fitness* yang dimiliki. Jika hal ini terjadi, algoritma genetik akan konvergen terlalu dini, padahal nilai yang dihasilkan belum tentu optimal secara keseluruhan. Dan inilah yang disebut dengan terjebak *pada local optimal*. Hal seperti ini bisa diatasi dengan menghindarkan kromosom super yang mendominasi populasi.

Fungsi evaluasi dilakukan dengan tiga tahap :

Tahap 1 Konversikan genotip menjadi fenotip. Bila kromosom berbentuk *bit string*, maka konversi dilakukan dengan mengkonversikan *binary* ke nilai real relatif

$x_k = (x_1^k, x_2^k)$, $k = 1,2,\ldots,pop\_size$ pop_size : ukuran populasi

Tahap 2 Evaluasi fungsi tujuan $f(x^k)$

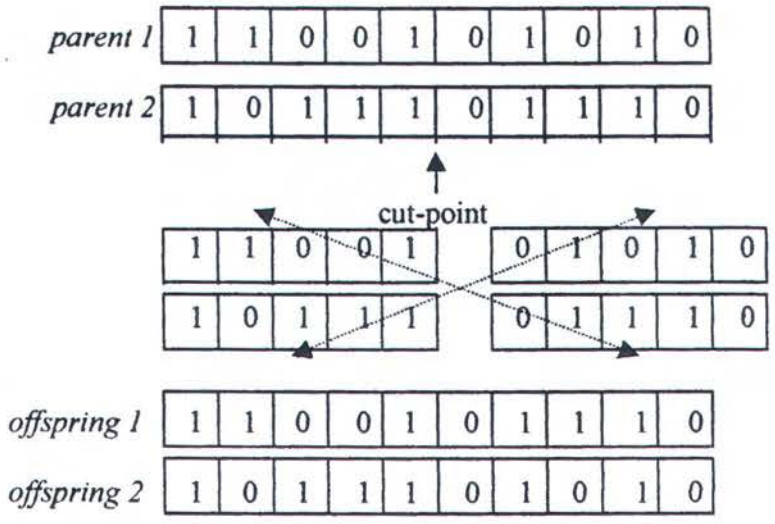
Tahap 3 Konversikan nilai fungsi tujuan menjadi nilai *fitness*

$Eval(v_k) = f(x^k), k = 1,2,\ldots,pop\_size$

III.2.3. Tukar silang

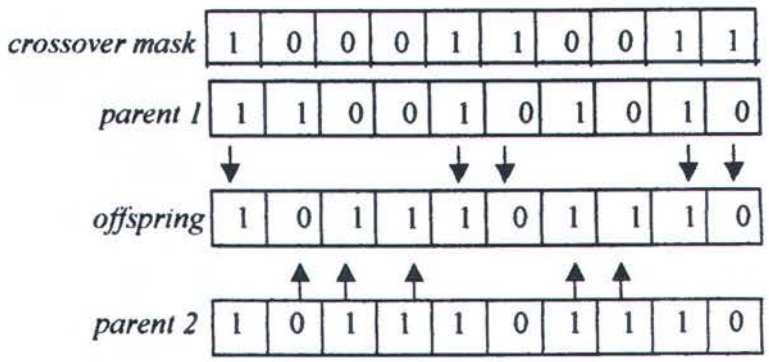
Operasi tukar silang dilakukan pada dua buah kromosom untuk menghasilkan kromosom *offspring* dengan mengkombinasikan gen – gen pada kedua kromosom. Setiap kromosom dibelah pada titik tertentu yang didapatkan secara acak disebut *cut-point*, sehingga pada kedudukan gen tertentu, kromosom ini terbelah menjadi dua bagian. Kemudian setiap bagian saling ditukar silang, sisi

kiri dari *cut-point* kromosom dikombinasikan dengan sisi kanan *cut-point* kromosom yang lain, dan sebaliknya. Dengan menggunakan metode tersebut, operasi tukar silang yang dilakukan disebut dengan ***one cut-point crossover***. Untuk lebih jelasnya diberikan ilustrasi sebagai berikut :



Gambar 3.6 Operasi tukar silang *one cut-point*

Selain metode tukar silang di atas, terdapat juga metode tukar silang uniform (***uniform crossover***). Gen – gen *offspring* dihasilkan oleh *parents* dengan bantuan *crossover mask*. Jika terdapat gen '1' pada *crossover mask*, gen dari *parent 1* yang diwariskan pada *offspring*. Jika gen '0' pada *crossover mask*, maka gen dari *parent 2* yang diwariskan. Metode ini dapat digambarkan sebagai berikut :



Gambar 3.7 Operasi tukar silang uniform

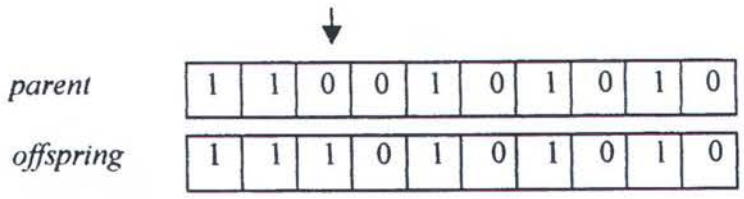
Pada proses tukar silang dikenal istilah tingkat tukar silang (*Crossover Rate*), yaitu besarnya kemungkinan suatu kromosom melakukan tukar silang dengan kromosom lain dalam setiap populasi. Atau dapat dikatakan bahwa dalam setiap populasi terdapat *Crossover Rate x PopSize* kromosom melakukan tukar silang.

```
procedure tukar silang;
begin
    k = 0;
    while (k <= pop_size) do
    begin
        rk = random [0, 1];
        if (rk < pc) then
        begin
            pilih 2 parent;
```

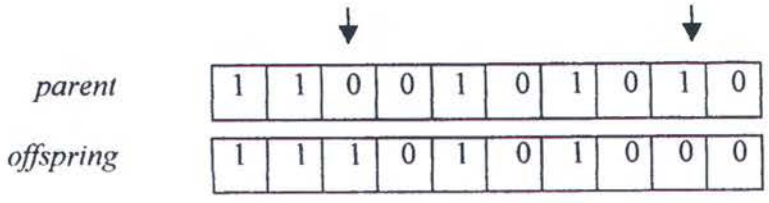
```
            proses tukar silang;
        end;
        k = k + 1;
    end;
end;
```

III.2.4. Mutasi

Mutasi adalah operasi membentuk *offspring* dengan mengubah susunan gen *parent*. Mutasi dapat dilakukan dengan metode inversi, yaitu terhadap suatu nilai gen dilakukan inversi dari '0' ke '1' atau sebaliknya. Metode yang lain adalah dengan menukar gen – gen yang terdapat pada kromosom yang disebut *exchange mutation*. Metode tersebut dapat dilihat pada gambar berikut :



Gambar 3.8 Operasi mutasi inversi



Gambar 3.9 Operasi mutasi *exchange*

Dalam proses mutasi dikenal istilah tingkat mutasi (*Mutation Rate*), yaitu besarnya kemungkinan setiap gen dari kromosom dalam suatu populasi untuk berubah. Atau dapat dikatakan bahwa terjadi mutasi sebanyak *Mutation Rate x PopSize x Chromosome Size* tiap generasi. *Chromosome Size* adalah panjang kromosom yaitu banyaknya gen.

```
procedure mutasi;
begin
    k = 0;
    while (k <= pop_size) do
    begin
        rk = random [0, 1];
        if (rk < pm) then
        begin
            pilih parent;
            proses mutasi;
        end;
        k = k + 1;
    end;
end;
```

III.2.5. Seleksi

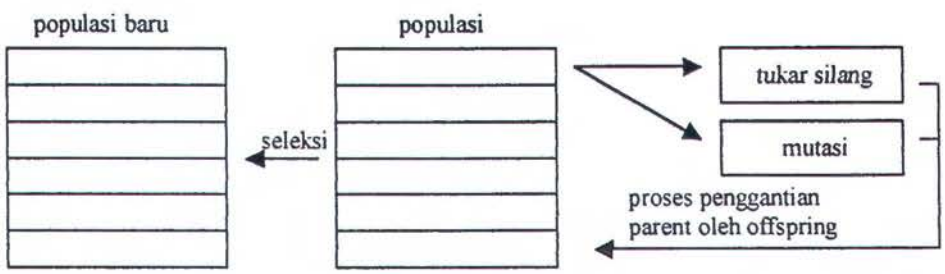
Ada tiga permaslahan penting yang sangat mempengaruhi seleksi yang harus diperhatikan, yaitu :

1. Daerah sampling
2. Mekanisme seleksi
3. Probabilitas seleksi.

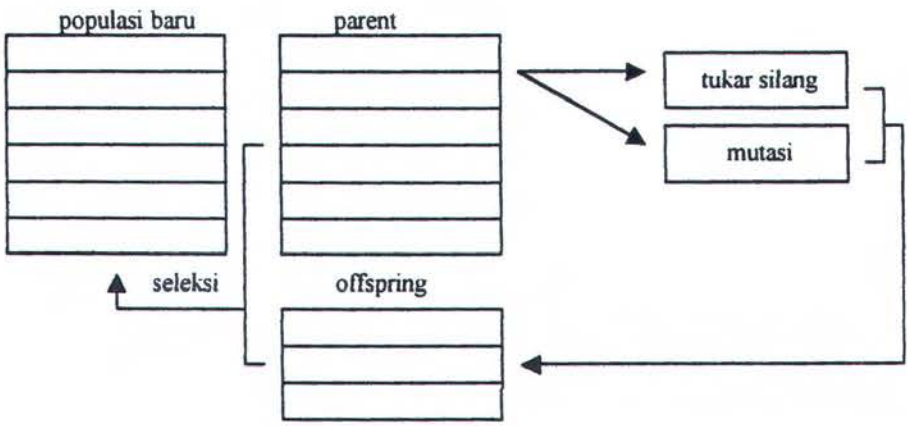
III.2.5.1. Daerah sampling

Seleksi menghasilkan generasi berikutnya. Kromosom – kromosom yang berada di dalamnya dapat berasal dari *parent* atau *offspring* atau gabungan keduanya. Asal kromosom inilah yang disebut dengan daerah sampling.

Daerah sampling dikatakan sebagai **daerah sampling yang tetap *(regular sampling space*)** bila *offspring* yang dihasilkan langsung menggantikan *parent* kemudian dikenakan seleksi. Namun bila *offspring* dan *parent* memiliki kemungkinan yang sama untuk bertahan hidup dan mengikuti seleksi untuk membentuk generasi baru, maka proses seleksi ini dikenakan pada **daerah sampling yang diperluas (*enlarged sampling space*)**. Berikut ini ilustrasi daerah sampling :



Gambar 3.10 Seleksi dengan daerah sampling yang tetap



Gambar 3.11 Seleksi dengan daerah sampling yang diperluas

III.2.5.2 Mekanisme sampling

Mekanisme sampling berguna untuk memilih kromosom dari daerah sampling yang dipertahankan hidup pada proses seleksi. Ada tiga jenis mekanisme sampling, yaitu :

- Sampling stokastik

  Tiap kromosom dipilih berdasarkan kemungkinannya bertahan hidup dengan memberikan probabilitas seleksi untuk setiap kromosom berdasarkan nilai fungsinya. Untuk kromosom k dengan nilai *fitness* f_k, probabilitas seleksi p_k adalah :

$$p_k = f_k \Big/ \sum_{j=1}^{pop\_size} f_j$$

Contoh sampling stokastik adalah *Roulette Wheel*

- Sampling deterministik

  Pemilihan kromosom berdasarkan kromosom terbaik dari daerah sampling. Metode elitis adalah contoh sampling deterministik, yaitu menentukan kromosom elit untuk diwariskan pada generasi berikutnya

- Sampling campuran

  Menggabungkan sampling stokastik dan sampling deterministik. Contohnya adalah seleksi turnamen. Dalam seleksi turnamen, sekelompok kromosom dipilih secara random untuk reproduksi.

III.2.5.3. Probabilitas seleksi

Probabilitas seleksi diberlakukan pada mekanisme sampling stokastik. Kromosom dengan nilai *fitness* tinggi akan mendominasi populasi. Untuk menghindari hal ini dilakukan penskalaan nilai *fitness* (*fitness scale*). Nilai *fitness* yang disesuaikan dinyatakan dengan f_k'. Hubungan f_k' dengan nilai *fitness* sebenarnya f_k adalah $f_k' = g(f_k)$.

BAB IV

PENERAPAN ALGORITMA GENETIK UNTUK PENJADUALAN JOB SHOP

Berikut akan dijelaskan representasi solusi, fungsi evaluasi, metode tukar silang, metode mutasi, dan seleksi pada algoritma genetik yang diterapkan untuk penjadualan *job shop*.

Permasalahan penjadualan *job shop* dapat direpresentasikan secara langsung (*direct approach*) maupun tak langsung (*indirect approach*). Pada representasi secara langsung, sebuah jadual dirupakan langsung dalam kromosom dan kinerja algoritma genetik ditujukan sebagai iterasi untuk memanipulasi kromosom tersebut sampai ditemukan jadual yang lebih baik/layak. Sedangkan pada representasi tak langsung, kromosom hanya merupakan perwujudan dari urutan aturan – aturan penugasan (*dispatching rules*) tertentu untuk mengacu pada urutan *job* yang diberikan. Algoritma genetik berguna untuk melakukan iterasi sampai ditemukan urutan aturan yang lebih baik dan kemudian sebuah jadual disusun dengan mengacu pada aturan tersebut.

IV.1. Representasi solusi

Pada kenyataannya walaupun telah banyak metode untuk menerapkan algoritma genetik pada penjadualan *job shop*, tetapi sangat sulit menentukan metode yang terbaik. Para peneliti juga saling melengkapi hasil penelitian lainnya dengan memperkenalkan metode – metode baru. Demikian halnya dengan

operator genetik. Tukar silang dan mutasi didesain sedemikian sehingga disesuaikan dengan metode representasi untuk menjaga daerah solusi agar tetap terdiri atas kromosom – kromosom yang *feasible*.

Salah satu metode representasi yang dapat digunakan adalah representasi berdasarkan *job* (*job based representation*). Representasi ini termasuk dalam *direct approach*. Reperesentasi ini menghasilkan urutan n *job* dan sebuah jadual yang tersusun dari urutan *job* tersebut. Setiap kromosom menyatakan urutan gen yang berisi *job* akan yang dijadualkan. Setiap gen dinyatakan dengan *integer*, yang menunjukkan *job* n. Penjadualan dimulai dari gen di urutan awal. Kemudian setiap gen yang menunjukkan urutan *job* tersebut dipetakan kepada urutan operasi yang terdapat dalam *job* yang bersangkutan. Jumlah operasi dalam setiap *job* sama dengan jumlah mesin (sesuai dengan karakteristik penjadualan *job shop*). Setiap operasi mempunyai hubungan korespondensi 1-1 dengan waktu pemrosesan dan mesin tempat pemrosesan masing – masing. Sehingga untuk setiap kromosom, bila setiap gen *job* teridentifikasi, algoritma mendeteksi urutan operasi dan mesin tempat operasi tersebut diproses serta lamanya waktu pemrosesan untuk selanjutnya diletakkan pada jadual menurut mesin masing – masing operasi. Dalam meletakkan setiap operasi dilakukan secara berurutan dari operasi 1, 2, sampai operasi n. Setelah seluruh operasi selesai dijadualkan, barulah algoritma berpindah pada gen berikutnya. Dan seterusnya sampai selesai.

Metode yang lain adalah representasi berdasarkan operasi (*operation based representation*). Reperesentasi ini menghasilkan urutan n *job* x m mesin dan sebuah jadual yang tersusun dari urutan *job* tersebut. Setiap kromosom

menyatakan urutan gen yang merepresentasikan operasi yang terdapat dalam suatu *job*. Namun perwujudannya berupa integer yang menunjukkan *job*. Setiap gen dinyatakan dengan *integer*, yang menunjukkan *job* n dan berulang sebanyak m mesin sesuai dengan urutan operasi pada job tersebut yang harus dikerjakan oleh mesin yang disediakan. Setiap gen dan operasi mempunyai hubungan korespondensi 1-1 dan setiap gen juga menyimpan informasi mesin dan waktu pemrosesan operasi/*job* yang bersangkutan.

Sehingga untuk setiap kromosom, bila setiap gen teridentifikasi, algoritma mendeteksi gen tersebut merepresentasikan operasi pada *job* tertentu dan mesin tempat operasi tersebut diproses serta lamanya waktu pemrosesan untuk selanjutnya diletakkan pada jadual. Deteksi yang dilakukan ini akan berguna pada saat menjalankan operator genetik agar pada generasi berikutnya tetap menjaga kromosom dengan urutan gen yang sesuai dengan urutan operasi – operasi pendahulu untuk setiap operasi pada *job* tertentu. Penjadualan dimulai dari gen 1, 2, sampai selesai [Cheng dan Gen, 1996].

Jika diberikan permasalahan tiga *job* dan tiga mesin seperti pada contoh yang telah diberikan, yaitu :

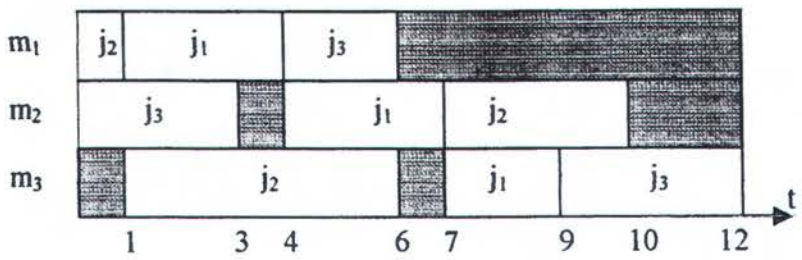
Tabel 4.1
Contoh permasalahan *job shop* 3x3

	Waktu pengerjaan				Mesin pemroses		
	Operasi				Operasi		
Job	1	2	3	Job	1	2	3
1	3	3	2	1	1	2	3
2	1	5	3	2	1	3	2
3	3	2	3	3	2	1	3

maka berikut ini dijelaskan representasi permasalahan tersebut menurut *job based* dan *operation based representation*.

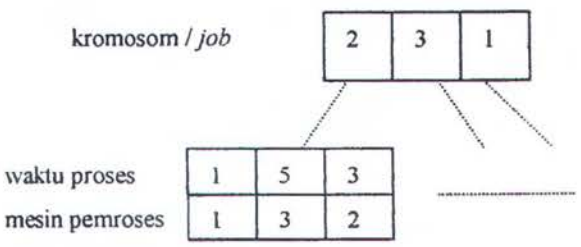
IV.1.1. Job based representation

Untuk permasalahan di atas, salah satu jadual yang *feasible* adalah sebagai berikut :



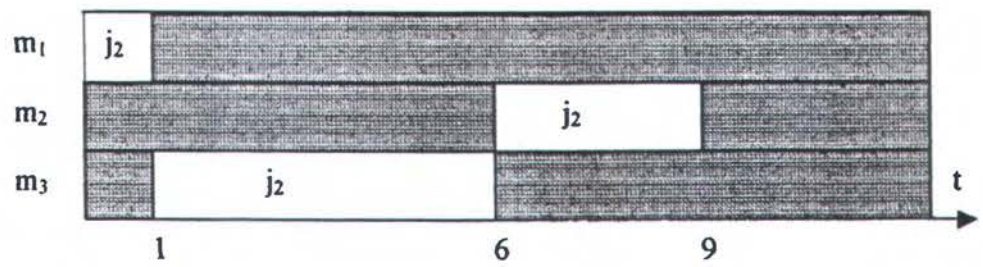
Gambar 4.1 Contoh solusi penjadualan *job* shop 3x3

Namun jika pada permasalahan tersebut digunakan algoritma genetik yang berawal dari himpunan solusi yang terdiri dari banyak kromosom, maka bila kromosom yang dihasilkan memiliki gen dengan urutan [2 3 1], menunjukkan bahwa penjadualan dimulai dari *job* 2, dilanjutkan *job* 3, dan *job* 1.



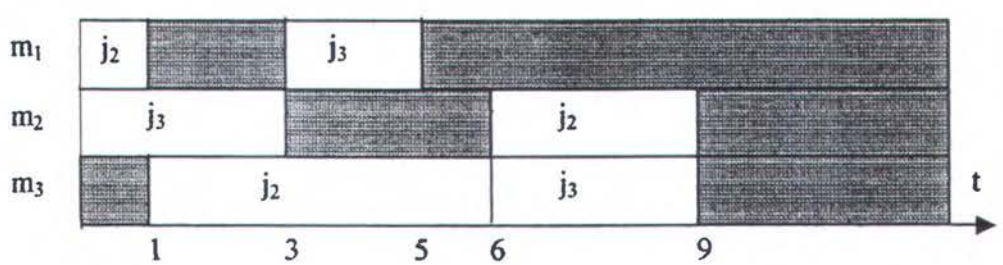
Gambar 4.2 Contoh *mapping* kromosom

Untuk *job* 2 yang terdiri dari urutan tiga operasi dengan lama waktu [1 5 3] yang dikerjakan pada urutan mesin [1 3 2], didapatkan penjadualan *job* 2 sebagai berikut :



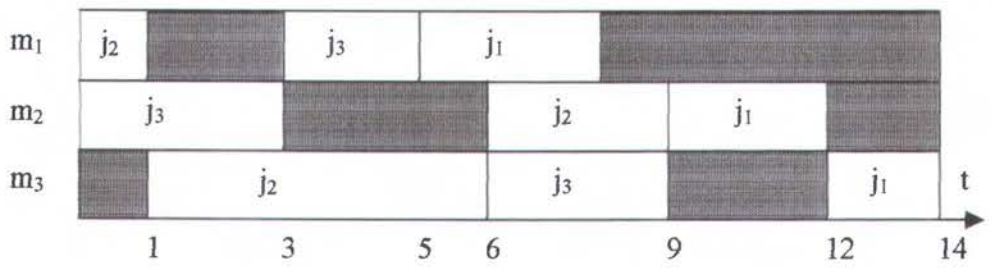
Gambar 4.3 Contoh *job-based representation* 3x3 : penjadualan gen 1

Kemudian *job* 3 yang terdiri dari urutan tiga operasi dengan lama waktu [3 2 3] yang dikerjakan pada urutan mesin [2 1 3], didapatkan penjadualan *job* 3 sebagai berikut :



Gambar 4.4 Contoh *job-based representation* 3x3 : penjadualan gen 2

Dan untuk job ke-1 yang terdiri dari urutan tiga operasi dengan lama waktu [3 3 2] pada urutan tiga mesin [1 2 3] didapatkan penjadualan job 2 sebagai berikut :



Gambar 4.5 Contoh *job-based representation* 3x3 : penjadualan gen 3

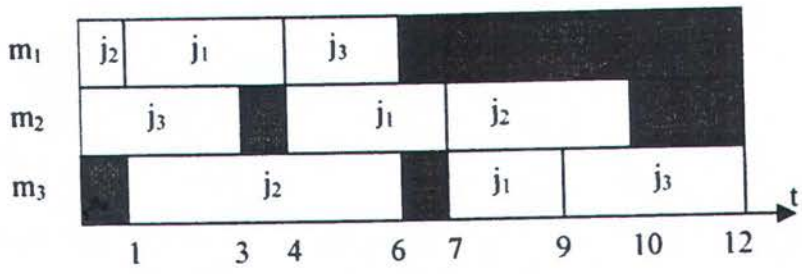
Ada hal yang perlu diperhatikan pada penyusunan jadual *job* ke 2, 3, dan seterusnya, yaitu bila operasi yang terdapat di dalamnya masih dapat diletakkan lebih awal dengan melompati operasi pada lain *job*, maka diizinkan untuk menggeser operasi tersebut pada waktu lebih awal. Hal ini sejalan dengan teori pembentukan jadual aktif.

IV.1.2. Operation based representation

Pada metode representasi ini setiap kromosom terdiri atas n x m gen yang merepresentasikan seluruh operasi yang harus dikerjakan. Setiap gen menunjukkan n job dan setiap job berulang sebanyak jumlah operasi, yaitu m mesin. Sehingga bila diberikan contoh permasalahan 3x3 seperti di atas, maka

salah satu kromosom yang mungkin adalah : [3 2 2 1 1 2 3 1 3]. Kromosom tersebut mengandung arti bahwa setiap *job* yang sama muncul menurut urutan operasi dengan operasi pendahuluannya. Misalkan *job* 3 muncul pada gen 1, 7, dan 9. Pada gen 1 adalah *job* 3 operasi 1, yaitu operasi dari *job* 3 yang dikerjakan pada mesin 2 dengan lama waktu pemrosesan = 3. Kemudian *job* 3 pada gen 7 adalah *job* 3 operasi 2 yang dikerjakan pada mesin 1 dengan lama waktu pemrosesan = 2. Dan *job* 3 pada gen 9 adalah operasi 3 yang dikerjakan pada mesin 3 dengan lama waktu pemrosesan = 3. Demikian seterusnya untuk *job* yang lain. Sehingga secara teknis pemrograman kromosom memuat nilai *job*, operasi, waktu, dan mesin. Jika dilihat dengan perspektif *job* dan operasi, kromosom di atas dapat dipahami sebagai berikut : [31 21 22 11 12 23 32 13 33].

Jika kromosom tersebut merupakan solusi akhir, maka penjadualan dilakukan mulai dari gen 1, 2, 3, 4, ... dan seterusnya sampai seluruh gen, sehingga jadual dibentuk dengan cara menjadualkan operasi – operasi sebagai berikut : *job* 3 operasi 1, *job* 2 operasi 1, *job* 2 operasi 2, *job* 1 operasi 1, *job* 1 operasi 2, *job* 2 operasi 3, *job* 3 operasi 2, *job* 1 operasi 3, *job* 3 operasi 3. Urutan tersebut tidak harus menjadi urutan waktu pada jadual kenyataannya. tetapi bila terdapat operasi yang berbeda *job* dan tidak mendahului operasi pendahuluannya yang dapat dijadualkan lebih awal, maka pilihan operasi tersebut diletakkan pada waktu yang lebih awal tersebut. Hal ini sesuai dengan teori pembentukan jadual aktif. Jadi, jadual yang terbentuk adalah sebagai berikut :



Gambar 4.6 Contoh penyelesaian penjadualan job shop dengan *opeation based representation*

IV.2. Fungsi evaluasi

Metode evaluasi yang sederhana untuk permasalahan penjadualan seperti penjadualan job shop, sering digunakan penghitungan inverse dari makespan. Sedangkan fungsi menghitung makespan didapat dari fungsi yang digunakan untuk meletakkan operasi pada jadual. Fungsi tersebut adalah :

$$\min t_n \quad \ldots\ldots\ldots\ldots\ldots\ldots \text{(i)}$$

batasan : $t_j - t_i \geq d_i$, $\quad (i, j) \in A \quad \ldots\ldots\ldots\ldots\ldots \text{(ii)}$

$$t_j - t_i \geq d_i \text{ atau } t_i - t_j \geq di\,, (i, j) \in E_k\,, k \in M \quad \ldots\ldots\ldots\ldots\ldots \text{(iii)}$$

$$t_j \geq 0 \quad \ldots\ldots\ldots\ldots \text{(iv)}$$

A menunjukkan himpunan (i, j) dengan i adalah operasi pada suatu *job* yang hanya dapat dikerjakan setelah operasi j

E_k adalah himpunan (i, j) dengan i adalah operasi yang dijadualkan pada mesin k dengan syarat tidak bersamaan dengan waktu operasi j

d_i = waktu pemrosesan operasi i

t_i = waktu dimulainya operasi i

(i) adalah fungsi tujuan,yaitu minimum makespan

(ii) menunjukkan urutan pengerjaan operasi pada suatu *job*

(iii) menunjukkan setiap mesin hanya mengerjakan sebuah *job* pada waktu yang sama

Jika T^k_{max} adalah *makespan*, maka fungsi evaluasi adalah :

$$v_k = 1 / T^k_{max}$$

IV.3. Tukar silang

Metode tukar silang yang digunakan adalah *order crossover* (OX). Sebagai pembanding digunakan *position based crossover* untuk *job based representation*. Sedangkan untuk *operation based representation* digunakan *order based* dan *precedence preservative crossover*.

IV.3.1. Order crossover

Order crossover diperkenalkan oleh Davis dan termasuk metode tukar silang yang baik untuk diterapkan pada permasalahan penjadualan. Berikut adalah algoritma metode *order crossover* :

Langkah 1 Pilih *substring* dari *parent* 1 secara random

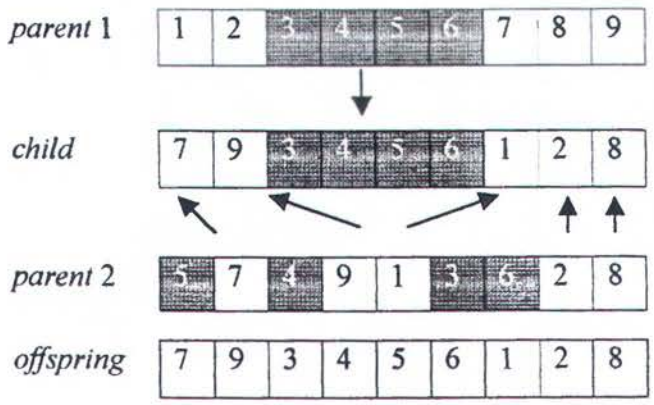
Langkah 2 Salin *substring* ke dalam *child* dengan posisi sama seperti pada *parent* 1

Langkah 3 Hapus gen pada *parent* 2 yang sama dengan *substring*

Langkah 4 Salin gen yang tersisa pada *parent* 2 ke *child* dengan urutan tetap dan terbentuklah *offspring*

Dengan cara yang sama membentuk *offspring* 2

Proses tersebut dapat digambarkan sebagai berikut :



Gambar 4.7 Proses *order crossover*

IV.3.2. Position based crossover

Position based crossover adalah alternatif metode tukar silang sebagai pembanding dan termasuk metode tukar silang yang baik untuk diterapkan pada permasalahan penjadualan. Berikut adalah algoritma metode *position based crossover* :

Langkah 1 Pilih gen dari *parent* 1 secara random

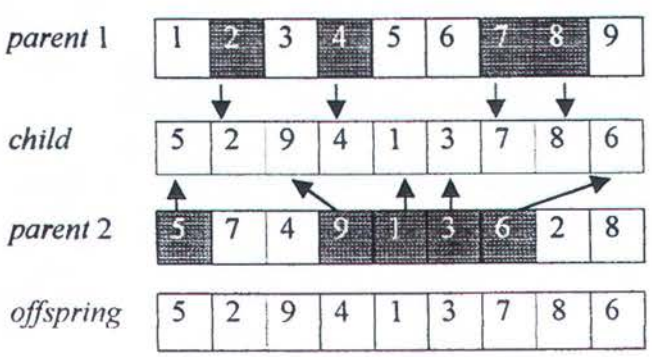
Langkah 2 Salin gen tersebut ke dalam *child* dengan posisi sama seperti pada *parent* 1

Langkah 3 Hapus gen yang sama dengan gen terpilih yang terdapat pada *parent* 2

Langkah 4 Lengkapilah gen yang tersisa pada *child* dengan gen dari *parent* 2 yang belum terdapat ke *child* dengan urutan tetap dan terbentuklah *offspring*

Dengan cara yang sama membentuk *offspring* 2

Proses tersebut dapat digambarkan sebagai berikut :



Gambar 4.8 Proses *position based crossover*

IV.3.3. Order based crossover

Sedangkan pada *operation based representation* menggunakan *order based crossover* dengan *one cut point* sebagai berikut :

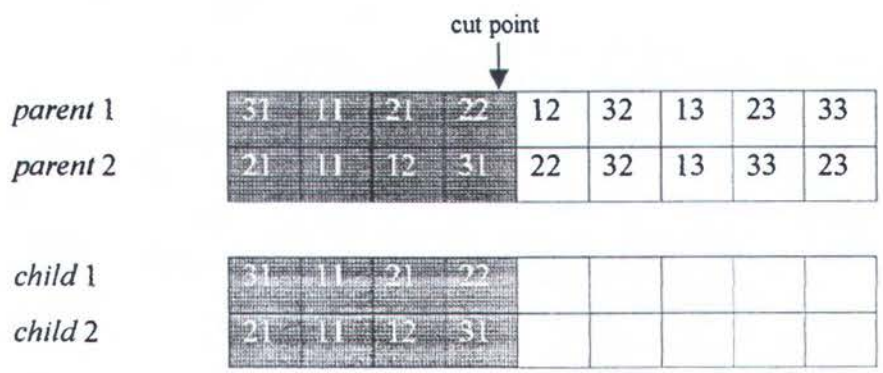
Langkah 1 Tentukan sebuah *cut point*, yaitu titik untuk membelah kedua *parent* menjadi dua bagian *substring*

Langkah 2 Salin *substring* sebelah kiri masing – masing ke dalam dua *child*

Langkah 3 Lengkapi *substring* sisa sebelah kanan pada *child* dengan mengambil gen secara silang dengan proses *leftmost*. Bila *child* berisi substring kiri dari *parent* 1, maka *substring* kanan diambil dari *parent* 2 dengan proses *leftmost* pada kromosom asal *parent* 2, dan sebaliknya

Langkah 4 Bila ditemukan gen yang berisi operasi yang belum ada pada *substring* kiri, salin gen tersebut dan seterusnya. Demikian juga untuk *child* yang lain dengan *parent* yang lainnya pula dan terbentuklah dua buah *offspring*.

Proses tersebut dapat digambarkan sebagai berikut :



Gambar 4.9 Proses *order based crossover*

offspring 1	31	11	21	22	12	32	13	33	23
offspring 2	21	11	12	31	22	32	13	23	33

Gambar 4.9 Proses *order based crossover* (lanjutan)

IV.3.4. Precedence preservative crossover

*Precedence preservative cross*over adalah alternatif lain dari metode tukar silang untuk *operation based representation.* Sesuai dengan namanya, metode ini dapat memelihara agar urutan operasi dengan operasi pendahulunya tidak berubah. Berikut adalah algoritma *precedence preservative crossover* :

Langkah 1 Pilih *parent* 1 dan *parent* 2

Langkah 2 Buatlah *vector crossover mask* dengan jumlah gen sama dengan parent dan hanya berisi nomer 1 dan 2 secara random

Langkah 3 Siapkan kromosom kosong untuk membentuk *offspring*

Langkah 4 Pilih gen secara *leftmost* baik dari *parent* 1 maupun *parent* 2 dengan bantuan *crossover mask* untuk diwariskan kepada *offspring*. Bila 1 pada *crossover mask*, maka gen diambil dari *parent 1* dan sebaliknya

Langkah 5 Untuk setiap gen yang sudah diwariskan, hapuslah gen tersebut baik dari *parent* 1 maupun *parent* 2

Langkah 6 Bila pilihan jatuh pada gen yang sudah diwariskan, maka proses *leftmost* diteruskan sampai pada gen yang belum terpilih.

Langkah 7 Ulangi langkah 4 sampai semua gen diwariskan. Dengan cara yang sama membentuk *offspring* 2

Proses tersebut dapat digambarkan sebagai berikut :

parent 1	31	11	21	22	12	32	13	33	23
parent 2	21	11	12	31	22	32	13	33	23
vector crossover mask	1	2	2	1	1	2	2	2	1
Offspring	31	21	11	22	12	32	13	33	23

Gambar 4.10 Proses *precedence preservative crossover*

IV.4. Mutasi

Metode mutasi yang digunakan pada *job based representation* adalah *reciprocal exchange* dan *insertion* sebagai pembanding. Kedua metode tersebut adalah metode mutasi yang umum digunakan untuk kromosom yang berupa permasalahan permutasi, seperti permasalahan penjadualan. Sedangkan pada *operation based representation* digunakan *order based mutation* untuk meneliti kembali letak gen dengan menyesuaikan gen yang berisi operasi pendahulunya.

IV.4.1. Reciprocal exchange

Metode ini menukar letak dua buah gen yang dipilih secara acak. Berikut adalah algoritma *reciprocal exchange* :

Langkah 1 Pilih dua gen secara acak

Langkah 2 Tukar kedua gen tersebut

Proses ini dapat digambarkan sebagai berikut :

parent	1	2	6	4	5	3	7	8	9
offspring	1	2	6	9	5	3	7	8	4

Gambar 4.11 Proses *reciprocal exchange*

IV.4.2. Insertion

Metode ini menyisipkan gen yang dipilih pada tempat yang ditentukan secara acak. Berikut adalah algoritma *insertion* :

Langkah 1 Pilih satu gen secara acak dan tentukan posisi *insertion*

Langkah 2 Sisipkan gen yang terpilih pada posisi *insertion*

Proses ini dapat digambarkan sebagai berikut :

parent	1	2	6	4	5	3	7	8	9
offspring	1	2	3	6	4	5	7	8	9

Gambar 4.12 Proses *insertion*

IV.4.3. Order based mutation

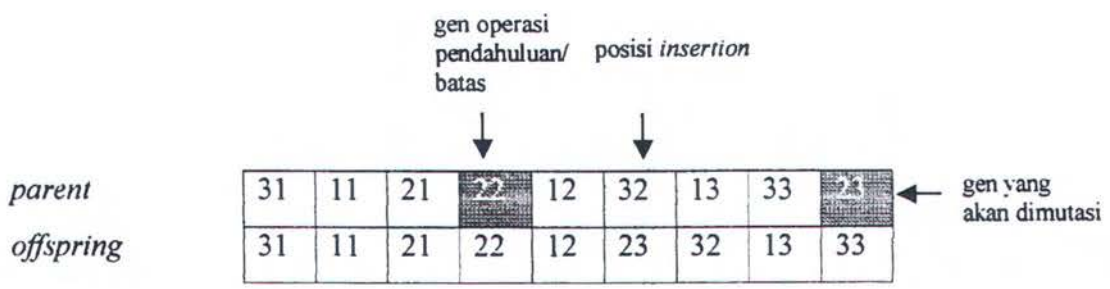
Metode ini seperti insertion, hanya berbeda pada saat menyisipkan gen terlebih dahulu harus memeriksa batas gen tersebut boleh disisipkan bila tidak mengubah urutan terhadap operasi pendahulunya.

Langkah 1 Pilih satu gen secara acak

Langkah 2 Periksa urutan gen berdasarkan operasi pendahulunya

Langkah 3 Sisipkan gen tersebut pada posisi *insertion* secara acak dengan batas gen dengan operasi pendahulunya

Proses ini dapat digambarkan sebagai berikut :



Gambar 4.13 Proses *order based mutation*

IV.5. Seleksi

Metode seleksi menggunakan *roulette wheel*. Berikut adalah langkah – langkah *roulette wheel* :

Langkah 1 Hitung nilai fungsi *fitness* masing – masing kromosom

$$Eval(K_i) = f(x), \qquad i = 1, 2, 3, \ldots, PopSize$$

Langkah 2 Hitung total nilai fungsi *fitness*

$$C = \sum_{i=1}^{PopSize} Eval(K_i) \qquad i = 1, 2, 3, \ldots, PopSize$$

Langkah 3 Hitung probabilitas seleksi untuk setiap kromosom

$$P_i = \frac{Eval(K_i)}{C} \qquad i = 1, 2, 3, \ldots, PopSize$$

Langkah 4 Hitung probabilitas seleksi kumulatif

$$Q_j = \sum_{i=1}^{j} P_i \qquad i = 1, 2, 3, \ldots, PopSize$$

Langkah 5 Bangkitkan bilangan random antara 0 dan 1

Langkah 6 Jika $r <= Q$, maka pilih kromosom K_i

Jika tidak, maka pilih kromosom K_j sedemikian rupa sehingga

$$Q_{j-1} < r <= Q_j$$

Pada proses seleksi juga dapat dilakukan *fitness scale*, yaitu cara untuk menghindari dominasi kromosom yang mempunyai nilai *fitness* tinggi. *Fitness scale* yang dibuat pada tugas akhir ini adalah *windowing* dan *exponential*.

Dengan *windowing* nilai *fitness* seluruh kromosom dikurangi dengan nilai *fitness* terkecil atau :

$$f'_k = f_k - f_w$$

Misalnya terdapat lima kromosom dengan nilai *fitness* : 0,255 (10%), 0,773 (29%) , 0,405 (15%), 0,928 (35%), 0,318 (11%). Maka setelah dilakukan *windowing* nilai *fitness* menjadi : 0,000 (0%), 0,518 (37%), 0,150 (11%), 0,673 (48%), 0,063 (4%). *Windowing* memperbesar probabilitas seleksi kromosom terkuat dan menghilangkan kromosom terlemah.

Dan *exponential* adalah menambahkan setiap nilai *fitness* dengan 1 kemudian dikuadratkan. Sehingga dengan data seperti di atas, maka setelah dilakukan *exponential* akan didapat nilai *fitness* : 1,575 (13%), 3,144 (26%), 1,974 (16%), 3,717 (31%), 1,737 (14%). *Exponential* memperbesar probabilitas seleksi dari kromosom yang relatif lemah.

BAB V

PERANCANGAN DAN IMPLEMENTASI PERANGKAT LUNAK

Perancangan perangkat lunak meliputi algoritma program, skema alur program dalam bentuk *data flow diagram*, rancangan data masukan, *class* yang terdapat pada perangkat lunak, dan data keluaran Sedangkan implementasi merupakan seluruh penerapan perancangan perangkat lunak. Data masukan dan keluaran sebagai alat untuk berinteraksi dengan pengguna (*user*).

V.1. Perancangan perangkat lunak

Berikut ini adalah rancangan perangkat lunak yang terdiri atas algoritma program disertai skema alur program dalam bentuk *data flow diagram*, data masukan, *class*, dan data keluaran.

V.1.1. Algoritma program

Algoritma program yang dibuat adalah :

Langkah 1 Memasukkan variabel penjadualan *job shop*, yaitu jumlah *job*, jumlah mesin, waktu pemrosesan *job*, dan mesin tempat *job* diproses

Langkah 2 Memasukkan parameter algoritma genetik, yaitu ukuran populasi, jumlah generasi, metode representasi, metode dan tingkat tukar silang, metode dan tingkat mutasi

Langkah 3 Inisialisasi populasi (generasi ke-1)

Langkah 4 Evaluasi nilai tiap kromosom

Langkah 5 Seleksi *roulette wheel*

Langkah 6 Tukar silang

Langkah 7 Mutasi

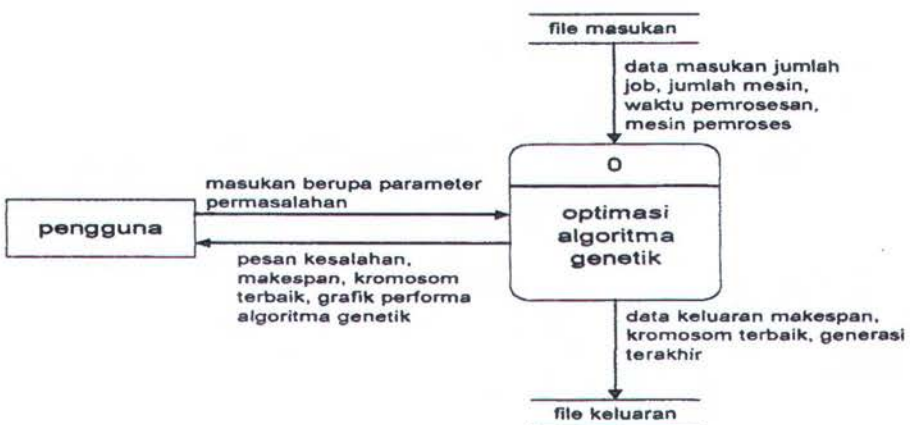
Langkah 8 Seleksi *elitism* untuk membentuk generasi berikutnya

Langkah 9 Mengulang mulai langkah 4 sampai sejumlah generasi

Untuk setiap generasi dihasilkan kromosom – kromosom pilihan pada proses seleksi untuk kemudian diwariskan kepada generasi selanjutnya

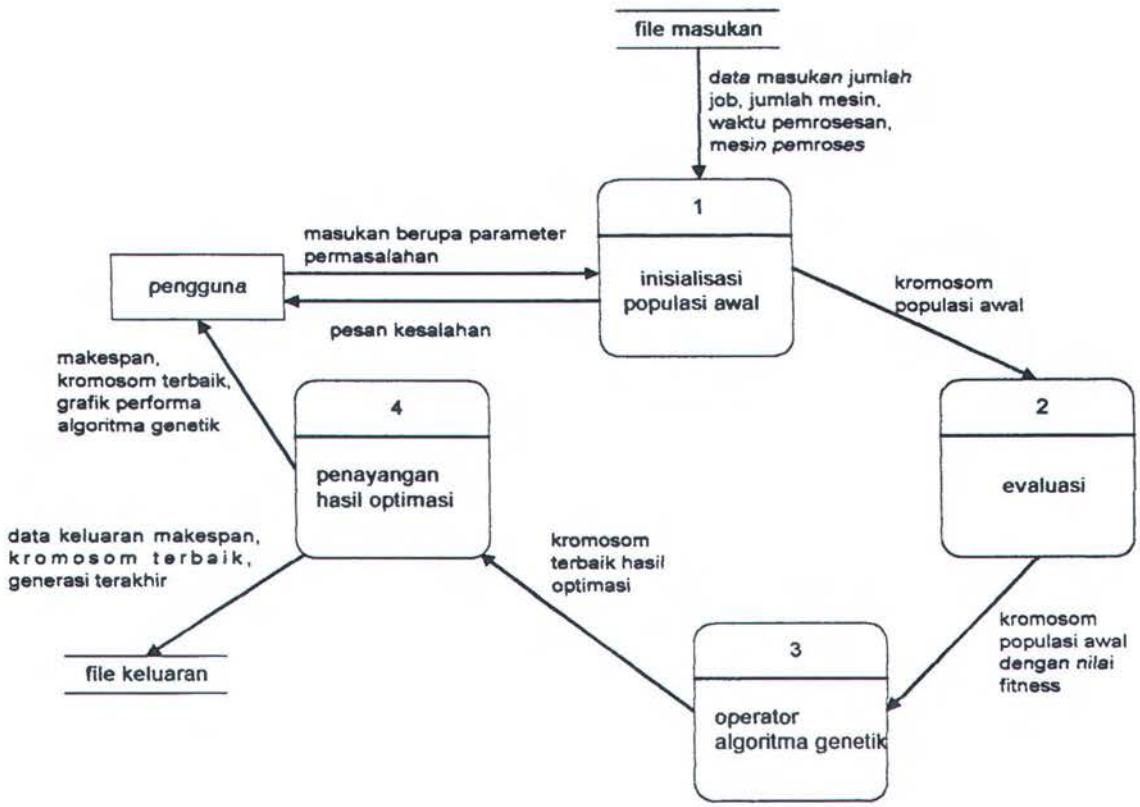
Langkah 10 Tampilkan keluaran berupa jadual, *makespan*, dan grafik performa algoritma genetik

Untuk lebih jelasnya dapat digambarkan sebagai *data flow diagram* yang dijelaskan berikut ini :



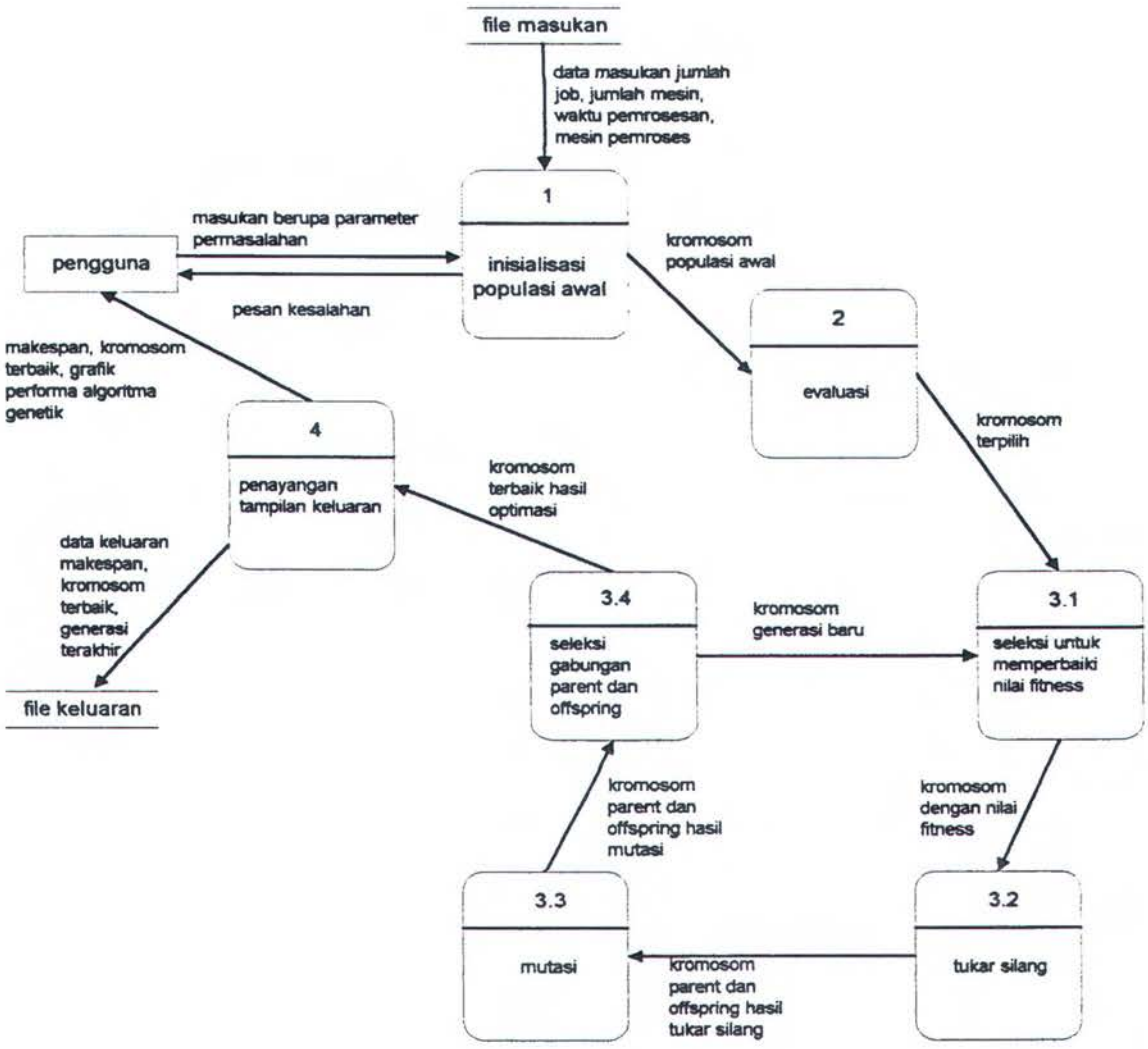
Gambar 5.1 *Data flow diagram* level 0

Optimasi dengan algoritma genetik dapat dijabarkan sebagai berikut :



Gambar 5.2 *Data flow diagram* level 1

Operator algoritma genetik dijabarkan kembali sebagai berikut :



Gambar 5.3 *Data flow diagram* level 2

V.1.2. Data masukan

Program yang akan dibuat menerima masukan langsung dari pengguna. Data masukan yang diperlukan sebagai permasalahan penjadualan adalah :

- Jumlah *job*, menunjukkan jumlah pekerjaan yang diproses
- Jumlah mesin, menunjukkan jumlah mesin yang dipakai
- Waktu pemrosesan, menunjukkan lama pengerjaan *job* di tiap mesin
- Data mesin yang mengerjakan operasi.

Dan parameter algoritma genetik yang diperlukan adalah :

- Jumlah generasi, menunjukkan jumlah maksimum generasi yang akan dibentuk selama proses algoritma genetik
- Ukuran populasi, menunjukkan jumlah kromosom pada suatu populasi
- Metode representasi, merupakan pilihan metode representasi
- Metode tukar silang, merupakan pilihan metode tukar silang
- *Crossover rate*, menunjukkan tingkat tukar silang
- Metode mutasi, merupakan pilihan metode mutasi
- *Mutation rate*, menunjukkan tingkat mutasi.

V.1.3. Struktur *class*

Untuk dapat menerapkan konsep di atas, maka digunakan struktur program menurut bahasa pemrograman tertentu. Perangkat lunak ini dibuat dengan bahasa pemrograman *Visual Basic 6.0*. Dalam *Visual Basic* obyek dapat berupa *class* yang berdiri sendiri. Di dalam obyek terdapat prosedur dan fungsi tersendiri

sesuai dengan kebutuhan yang dapat digunakan hanya di dalam obyek itu sendiri (*private*) maupun di luar obyek yang bersangkutan (*public*).

Rancangan perangkat lunak ini terdiri atas empat buah *class*, yaitu :

- *Class CJobshop*, merepresentasikan permasalahan *job shop*
- *Class CGenetic*, merepresentasikan algoritma genetik dengan metode representasi *job based representation*
- *Class CGenetic2*, merepresentasikan algoritma genetik dengan metode representasi *operation based representation*
- *Class CSpt*, merepresentasikan algoritma *shortest processing time*.

Berikut adalah keterangan tentang *class* :

1. ***Class CJobshop***, memiliki *interface* sebagai berikut :
   - Jumlah *job*, menunjukkan jumlah pekerjaan dan dinyatakan dalam integer
   - Jumlah mesin, menunjukkan jumlah mesin dan dinyatakan dalam integer
   - Waktu proses menunjukkan waktu pengerjaan pekerjaan i pada mesin j dan dinyatakan dalam array integer satu dimensi
   - Mesin j pemroses *job* i dan dinyatakan dengan array integer satu dimensi
   - Fungsi *CalculateMSpan*, merupakan fungsi untuk menghitung *makespan*.

     Public Function CalculateMSpan(C As Chromosome, JumlahMesin As Integer, JumlahJob As Integer, Jadual() As Bar) As Integer
     ... ...
     End Function
     Public Function CalculateMSpan1(C As Chromosome1, JumlahMesin As Integer, JumlahJob As Integer, Jadual1() As Bar1) As Integer
     ... ...
     End Function

2. ***Class CGenetic***, memiliki *interface* sebagai berikut :

- Inisialisasi populasi, menunjukkan cara menentukan populasi awal dan diimplementasikan secara random

```
Private Sub PopInit()
... ... ... ... ... ...
    IdxJob = Int((Rnd * JumlahJob) - 1)
... ... ... ... ... ...
End Sub
```

- Ukuran populasi atau PopSize, menunjukkan banyaknya individu dalam sebuah populasi dan dinyatakan dalam integer
- Jumlah generasi dinyatakan dalam integer
- *Crossover rate*, menunjukkan tingkat tukar silang dan dinyatakan dalam *single*
- Pilihan metode tukar silang, diimplementasikan dengan *select case*

```
Private Sub crossover(K1 As Chromosome, K2 As Chromosome,
Output1
    As Chromosome, Output2 As Chromosome)
... ... ... ... ... ...
Select Case CrossMethod
  Case Orderc
  ... ... ... ... ... ...
  Case Positionbased
  ... ... ... ... ... ...
End Select
End Sub
```

- *Mutation rate*, menunjukkan tingkat mutasi, dinyatakan dalam *single*
- Pilihan metode mutasi, diimplementasikan dengan *select case*

```
Private Function mutation(k as Chromosome, i1 As Integer)
... ... ... ... ... ...
Select Case MutMethod
  Case Reciprocal
... ... ... ... ... ...
```

```
Case Insertion
... ... ... ... ... ...
End Select
End Sub
```

- RunGA, merupakan fungsi untuk melakukan optimasi dengan algoritma genetik.

3. ***Class CGenetic2***, memiliki *interface* sebagai berikut :

- Inisialisasi populasi, menunjukkan cara menentukan populasi awal dan diimplementasikan secara random

```
Private Sub PopInit()
... ... ... ... ... ...
indexjob = Int(Rnd * JumlahJob + 1)
... ... ... ... ... ...
End Sub
```

- Ukuran populasi atau PopSize, menunjukkan banyaknya individu dalam sebuah populasi dan dinyatakan dalam integer
- Jumlah generasi dinyatakan dalam integer
- *Crossover rate*, menunjukkan tingkat tukar silang dan dinyatakan dalam *single*
- Pilihan metode tukar silang, diimplementasikan dengan *select case*

```
Private Sub crossover(K1 As Chromosome1, K2 As Chromosome1,
        Output1 As Chromosome1, Output2 As Chromosome1)
... ... ... ... ... ...
Select Case CrossMethod
Case Orderbased
... ... ... ... ... ...
Case Precedence
... ... ... ... ... ...
End Select
End Sub
```

- *Mutation rate*, menunjukkan tingkat mutasi, dinyatakan dalam *single*
- Metode mutasi diimplementasikan hanya satu metode

```
Private Sub mutation(k As Chromosome1, i1 As Integer)
... ... ... ... ... ...
... ... ... ... ... ...
... ... ... ... ... ...
End Sub
```

- RunGA, merupakan fungsi untuk melakukan optimasi dengan algoritma genetik.

4. ***Class CSpt***, memiliki *interface* sebagai berikut :
   - Jumlah *job*, menunjukkan jumlah pekerjaan dan dinyatakan dalam integer
   - Jumlah mesin, menunjukkan jumlah mesin dan dinyatakan dalam integer
   - Waktu proses menunjukkan waktu pengerjaan pekerjaan i pada mesin j dan dinyatakan dalam array integer satu dimensi
   - Mesin j pemroses *job* i dan dinyatakan dengan array integer satu dimensi
   - Fungsi *CalculateMspan*, merupakan fungsi untuk menghitung *makespan.*

```
Public Function CalculateMSpan(S( ) As Integer) As Integer
... ... ... ... ... ...
End Function
```

V.1.3. Data keluaran

Data keluaran berupa hasil optimasi yang terdiri atas :

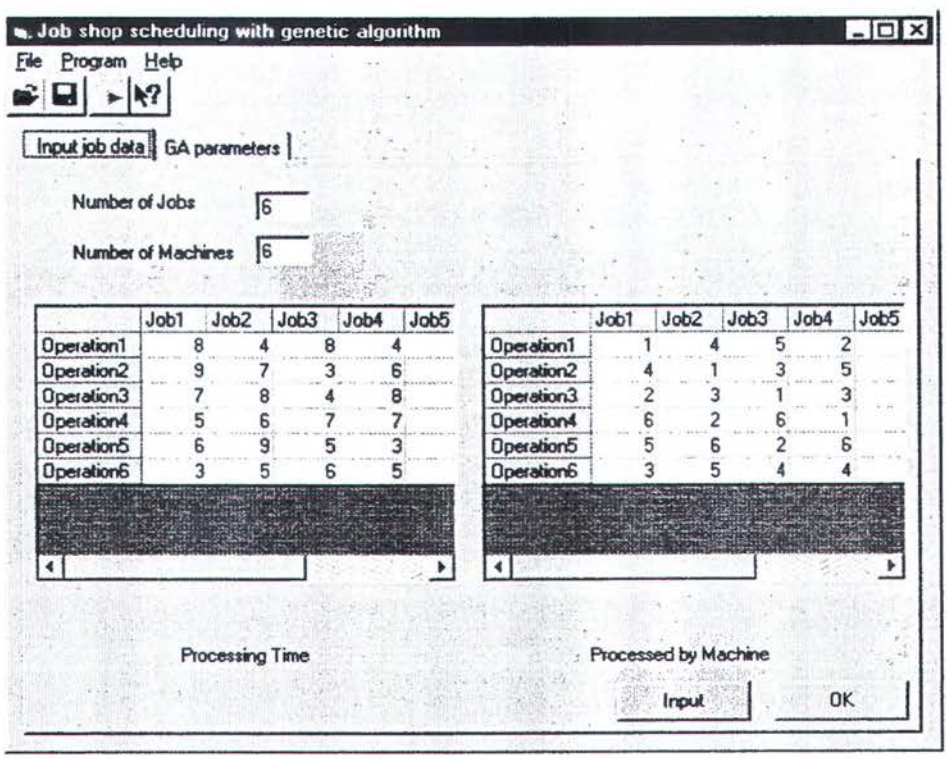
- Jadual dengan keterangan *makespan* dan kromosom
- Grafik kinerja algoritma genetik
- Keterangan detil permasalahan yang diselesaikan.

V.2. Implementasi perangkat lunak

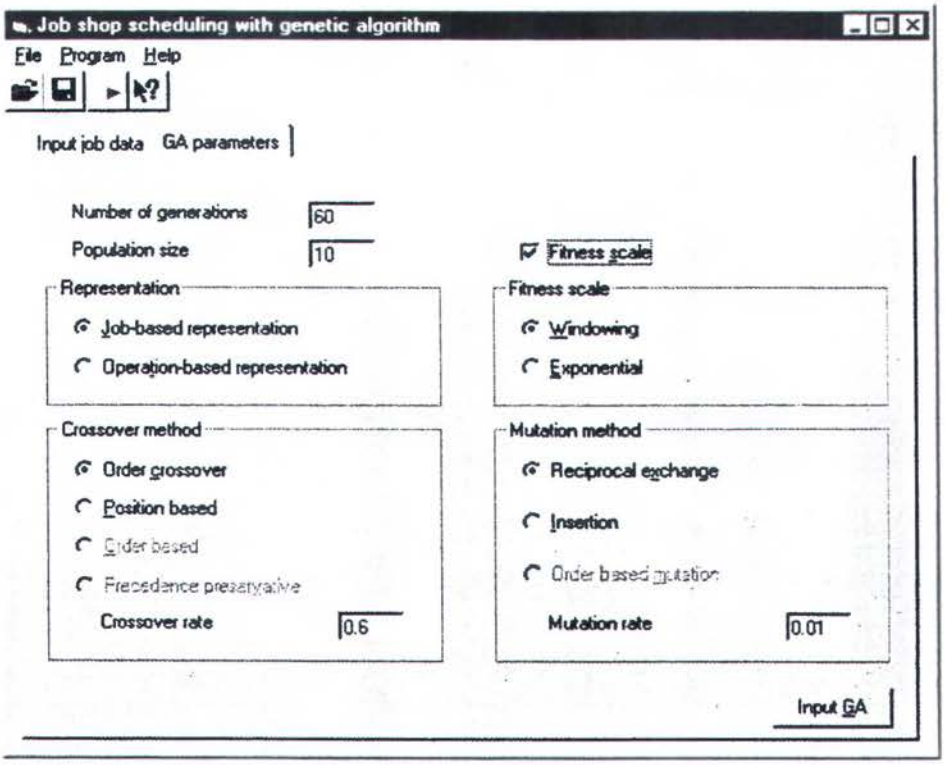
Berikut ini adalah implementasi dari rancangan perangkat lunak yang sudah dijelaskan di atas.

V.2.1. Tampilan masukan

Untuk berinteraksi dengan *user*, maka perlu dibuat tampilan masukan untuk menerima data masukan yang dilakukan secara langsung. Data yang dimasukkan seperti telah disebutkan sebelumnya dan tampilan masukannya adalah :



Gambar 5.4 Tampilan masukan data *job*



Gambar 5.5 Tampilan masukan parameter algoritma genetik

V.2.2. Implementasi class

Berikut ini adalah implementasi dari *class* yang ada pada rancangan yang telah dijelaskan sebelumnya :

Deklarasi kromosom dalam bentuk record

Kromosom untuk *job based representation*

```
Public Type Chromosome
  Job()      As Integer
  Fitness    As Single
  Rfitness   As Single
```

```
  Cfitness        As Single
End Type
```

Kromosom untuk *operation based representation*

```
Public Type Chromosome1
  gens()        As Gogen
  Fitness1      As Single
  Rfitness1     As Single
  Cfitness1     As Single
End Type

Public Type Gogen
  gojob As Integer
  goop As Integer
  gotime As Integer
  gomesin As Integer
End Type
```

Private Function CalculateMSpan* untuk menghitung *makespan* pada *job based representation*. *Function* ini terdapat dalam *class CJobshop

```
ReDim Jadual(JumlahMesin, JumlahJob)
CalculateMSpan = 0
For i = 1 To JumlahJob
  Mulai = 0
  For j = 1 To JumlahMesin
    Time = ProTime(C.Job(i), j)
    Mesin = ProMesin(C.Job(i), j)
    Jadual(Mesin, i).IdxJob = C.Job(i)
    Jadual(Mesin, i).Start = Mulai
    Ptr = i
    For k = 1 To i – 1
      If Jadual(Mesin, Ptr).Start + Time <= Jadual(Mesin, k).Start Then
        Temp = Jadual(Mesin, Ptr)
        For l = Ptr To k + 1 Step -1
        Jadual(Mesin, l) = Jadual(Mesin, l - 1)
        Next
        Jadual(Mesin, k) = Temp
        Ptr = k
        Exit For
```

```
        ElseIf Jadual(Mesin, Ptr).Start < Jadual(Mesin, k).Finish Then
          Jadual(Mesin, Ptr).Start = Jadual(Mesin, k).Finish
        End If
      Next
      Jadual(Mesin, Ptr).Finish = Jadual(Mesin, Ptr).Start + Time
      Mulai = Jadual(Mesin, Ptr).Finish
      If Mulai > CalculateMSpan Then CalculateMSpan = Mulai
    Next
  Next
```

Private Function CalculateMSpan1 **untuk menghitung** ***makespan*** **pada** ***operation based representation. Function*** **ini terdapat dalam** ***class CJobshop***

```
  For i = 1 To leng
    Time = C.gens(i).gotime
    Mesin = C.gens(i).gomesin
    IndexMesin(Mesin) = IndexMesin(Mesin) + 1
    Ptr = IndexMesin(Mesin)
    Jadual1(Mesin, Ptr).IdxJob1 = C.gens(i).gojob
    Jadual1(Mesin, Ptr).Start1 = FinishJob(C.gens(i).gojob) 'Mulai
     For j = 1 To Ptr - 1
      If (Jadual1(Mesin, Ptr).Start1 >= Jadual1(Mesin, j).Start1) And
          (Jadual1(Mesin, Ptr).Start1 < Jadual1(Mesin, j).Finish1) Then
        Jadual1(Mesin, Ptr).Start1 = Jadual1(Mesin, j).Finish1
        Exit For
      End If
     Next

     For k = j To Ptr - 1
       If Jadual1(Mesin, Ptr).Start1 + Time <= Jadual1(Mesin, k).Start1
      Then
          Temp = Jadual1(Mesin, Ptr)
          For l = Ptr To k + 1 Step -1
            Jadual1(Mesin, l) = Jadual1(Mesin, l - 1)
          Next
          Jadual1(Mesin, k) = Temp
          Exit For
      ElseIf Jadual1(Mesin, Ptr).Start1 < Jadual1(Mesin, k).Finish1 Then
          Jadual1(Mesin, Ptr).Start1 = Jadual1(Mesin, k).Finish1
      End If
     Next

     Jadual1(Mesin, Ptr).Finish1 = Jadual1(Mesin, Ptr).Start1 + Time
     FinishJob(C.gens(i).gojob) = Jadual1(Mesin, Ptr).Finish1
     If FinishJob(C.gens(i).gojob) > CalculateMSpan1 Then
```

```
    CalculateMSpan1 = FinishJob(Cgens(i).gojob)
  Next
```

Private Sub PopInit **untuk inisialisasi populasi awal pada** ***job based representation.*** **Prosedur ini terdapat dalam** ***class CGenetic***

```
ReDim dtPopulasi(PopSize)
  For i = 1 To PopSize
    ReDim dtPopulasi(i).Job(JumlahJob)
    For Idx = 1 To JumlahJob

      'Set Randomize
       Do
         IdxJob = Int(JumlahJob * Rnd + 1)
         Found = False
         For j = 1 To Idx - 1
           Found = (IdxJob = dtPopulasi(i).Job(j))
           If Found Then Exit For
         Next
       Loop Until Not Found
       dtPopulasi(i).Job(Idx) = IdxJob
  Next
Next
```

Private Sub PopInit **pada** ***operation based representation.*** **Prosedur ini terdapat dalam** ***class CGenetic2***

```
For i = 1 To PopSize
  ReDim dtPopulasi1(i).gens(leng)
  jml = 1
  h = 1
  'Set randomize
   Do
    indexjob = Int(Rnd * JumlahJob + 1)
    Y = 1
    For k = jml - 1 To 1 Step -1
     foundjob = (dtPopulasi1(i).gens(k).gojob = indexjob)
     If foundjob Then
       h = dtPopulasi1(i).gens(k).goop
       If h = JumlahMesin Then
         Y = Y + 1
         jml = jml - 1
         Exit For
       Else
```

```
        dtPopulasi1(i).gens(jml).gojob = indexjob
        dtPopulasi1(i).gens(jml).goop = h + 1
        jml = jml - 1
        Y = Y + 1
        Exit For
       End If
      End If
     Next
     If jml = 1 Then
      dtPopulasi1(i).gens(jml).gojob = indexjob
      h = 1
      dtPopulasi1(i).gens(jml).goop = 1
      jml = jml + 1
     Else
      If Y = 1 Then
       dtPopulasi1(i).gens(jml).gojob = indexjob
       dtPopulasi1(i).gens(jml).goop = 1
       jml = jml + 1
      End If
     End If
    Loop Until jml > leng
  Next
```

Private Sub crossover **untuk melakukan proses tukar silang pada** ***job based representation.*** **Prosedur ini terdapat dalam** ***class CGenetic***

```
Select Case CrossMethod
 Case Orderc
  Do
    r1 = Int(JumlahJob * Rnd + 1)
    r2 = Int(JumlahJob * Rnd + 1)
  Loop Until (r2 >= r1) And (r2 - r1 + 1 < JumlahJob)
  jml = r2 - r1 + 1
  s = jml
  idex = r1

  For i = r1 To r2
    Output1.Job(i) = p1.Job(i)
  Next

  ReDim tamp(s) As Integer

  For i = 1 To jml
    tamp(i) = Output1.Job(idex)
    idex = idex + 1
```

```
Next

For i = 1 To JumlahJob
   For j = 1 To jml
      If p2.Job(i) = tamp(j) Then p2.Job(i) = 0
   Next
Next

For i = 1 To JumlahJob
   idx = 1
   If p2.Job(i) <> 0 Then
     For idx = 1 To JumlahJob
        If Output1.Job(idx) = 0 Then
           Output1.Job(idx) = p2.Job(i)
           Exit For
        End If
     Next
  End If
Next

For i = r1 To r2
   Output2.Job(i) = p4.Job(i)
Next
idex = r1
For i = 1 To jml
   tamp(i) = Output2.Job(idex)
   idex = idex + 1
Next

For i = 1 To JumlahJob
   For j = 1 To jml
      If p3.Job(i) = tamp(j) Then p3.Job(i) = 0
   Next
Next
For i = 1 To JumlahJob
    idx = 1
    If p3.Job(i) <> 0 Then
      For idx = 1 To JumlahJob
         If Output2.Job(idx) = 0 Then
           Output2.Job(idx) = p3.Job(i)
           Exit For
        End If
      Next
   End If
 Next
```

```
Case Positionbased
 Do
  jumlahpick = Int(JumlahJob * Rnd + 1)
 Loop Until (jumlahpick <= (JumlahJob - 2))

 ReDim pickitem(jumlahpick)

 For i = 1 To jumlahpick
 'Set Randomize
   Do
      pidx = Int(JumlahJob * Rnd + 1)
     Found = False
     For j = 1 To jumlahpick - 1
        Found = (pidx = pickitem(j))
        If Found Then Exit For
     Next
   Loop Until Not Found
   pickitem(i) = pidx
   Output1.Job(pidx) = p1.Job(pidx)
   Output2.Job(pidx) = p4.Job(pidx)
 Next
 For i = 1 To JumlahJob
   For j = 1 To jumlahpick
     If p2.Job(i) = Output1.Job(pickitem(j)) Then p2.Job(i) = 0
     If p3.Job(i) = Output2.Job(pickitem(j)) Then p3.Job(i) = 0
   Next
 Next

 i = i + 1

 For i = 1 To (JumlahJob - jumlahpick)
   j = 1
   k = 1
   m = 1
   n = 1
   idx = p2.Job(j)
   idx1 = p3.Job(k)
  While idx = 0
      j = j + 1
      idx = p2.Job(j)
  Wend
  While idx1 = 0
      k = k + 1
      idx1 = p3.Job(k)
  Wend
  idex = Output1.Job(m)
```

```
idex1 = Output2.Job(n)
While idex <> 0
    m = m - 1
    idex = Output1.Job(m)
Wend
While idex1 <> 0
    n = n - 1
    idex1 = Output2.Job(n)
Wend
Output1.Job(m) = idx
Output2.Job(n) = idx1
For w = 1 To JumlahJob
    If p1.Job(w) = idx Then p1.Job(w) = 0
    If p2.Job(w) = idx Then p2.Job(w) = 0
    If p3.Job(w) = idx1 Then p3.Job(w) = 0
    If p4.Job(w) = idx1 Then p4.Job(w) = 0
Next
Next

End Select
End Sub
```

Private Sub crossover **pada** ***operation based representation.*** **Prosedur ini terdapat dalam** ***class CGenetic2***

```
Select Case CrossMethod

Case Orderbased
  cekgojob = True
  Do
    Do
      cutpick = Int(Rnd * leng + 1)
    Loop Until (cutpick <= (leng - 2))
    beda = leng - cutpick
    j = 1
    ReDim pickgen(beda)
    For l = cutpick + 1 To leng
      pickgen(j).gojob = p1.gens(l).gojob
      If j > 1 Then
        If pickgen(j).gojob = pickgen(j - 1).gojob Then
          If j = beda Then
            cekgojob = True
            Exit For
          End If
```

```
        j = j - 1
      Else
        cekgojob = False
        Exit For
      End If
    Else
      j = j - 1
    End If
  Next
Loop Until (cekgojob = False)


For i = 1 To cutpick
  Output1.gens(i) = p1.gens(i)
  Output2.gens(i) = p2.gens(i)
Next

For i = 1 To cutpick
  For j = 1 To leng
    If ((p2.gens(j).gojob = Output1.gens(i).gojob) And
       (p2.gens(j).goop = Output1.gens(i).goop)) Then
        p2.gens(j).gojob = 0
        p2.gens(j).goop = 0
    End If
  Next
  For q = 1 To leng
    If ((p1.gens(q).gojob = Output2.gens(i).gojob) And
      (p1.gens(q).goop = Output2.gens(i).goop)) Then
      p1.gens(q).gojob = 0
      p1.gens(q).goop = 0
    End If
  Next
Next

For fillgen = cutpick + 1 To leng
  s = 1
  idx2 = p1.gens(s)
  While ((idx2.gojob = 0) And (idx2.goop = 0))
    s = s + 1
    idx2 = p1.gens(s)
  Wend
  Output2.gens(fillgen) = idx2
  p1.gens(s).gojob = 0
  p1.gens(s).goop = 0
Next
```

```
For fillgen = cutpick - 1 To leng
   s = 1
   idx2to = p2.gens(s)
   While ((idx2to.gojob = 0) And (idx2to.goop = 0))
      s = s + 1
      idx2to = p2.gens(s)
   Wend
   Output1.gens(fillgen) = idx2to
   p2.gens(s).gojob = 0
   p2.gens(s).goop = 0
Next
```

Case Precedence

```
   For i = 1 To leng
      Select Case Int(2 * Rnd + 1)
       Case 1
          j = 1
          k = 1
          idx2.gojob = p1.gens(j).gojob
          idx3.goop = p1.gens(j).goop
          idx2to.gojob = p4.gens(k).gojob
          idx3to.goop = p4.gens(k).goop
          While ((idx2.gojob = 0) And (idx3.goop = 0))
             j = j + 1
             idx2.gojob = p1.gens(j).gojob
             idx3.goop = p1.gens(j).goop
          Wend
          While ((idx2to.gojob = 0) And (idx3to.goop = 0))
             k = k + 1
             idx2to.gojob = p4.gens(k).gojob
             idx3to.goop = p4.gens(k).goop
          Wend
          idex2.gojob = Output1.gens(j).gojob
          idex3.goop = Output1.gens(j).goop
          idex2to.gojob = Output2.gens(k).gojob
          idex3to.goop = Output2.gens(k).goop
          While ((idex2.gojob <> 0) And (idex3.goop <> 0))
             j = j + 1
             idex2.gojob = Output1.gens(j).gojob
             idex3.goop = Output1.gens(j).goop
          Wend
          While ((idex2to.gojob <> 0) And (idex3to.goop <> 0))
             k = k + 1
             idex2to.gojob = Output2.gens(k).gojob
             idex3to.goop = Output2.gens(k).goop
```

```
    Wend
    Output1.gens(j).gojob = idx2.gojob
    Output1.gens(j).goop = idx3.goop
    Output2.gens(k).gojob = idx2to.gojob
    Output2.gens(k).goop = idx3to.goop
    For w = 1 To leng
   If p1.gens(w).gojob = idx2.gojob Then p1.gens(w).gojob = 0
    If p2.gens(w).gojob = idx2.gojob Then p2.gens(w).gojob = 0
   If p3.gens(w).gojob = idx2to.gojob Then p3.gens(w).gojob = 0
   If p4.gens(w).gojob = idx2to.gojob Then p4.gens(w).gojob = 0
   If p1.gens(w).goop = idx3.goop Then p1.gens(w).goop = 0
   If p2.gens(w).goop = idx3.goop Then p2.gens(w).goop = 0
   If p3.gens(w).goop = idx3to.goop Then p3.gens(w).goop = 0
   If p4.gens(w).goop = idx3to.goop Then p4.gens(w).goop = 0
  Next
```

Case 2

```
  j = 1
  k = 1
  idx2.gojob = p2.gens(j).gojob
  idx3.goop = p2.gens(j).goop
  idx2to.gojob = p3.gens(k).gojob
  idx3to.goop = p3.gens(k).goop
  While ((idx2.gojob = 0) And (idx3.goop = 0))
     j = j + 1
     idx2.gojob = p2.gens(j).gojob
     idx3.goop = p2.gens(j).goop
  Wend
  While ((idx2to.gojob = 0) And (idx3to.goop = 0))
     k = k + 1
     idx2to.gojob = p3.gens(k).gojob
     idx3to.goop = p3.gens(k).goop
  Wend
  idex2.gojob = Output1.gens(j).gojob
  idex3.goop = Output1.gens(j).goop
  idex2to.gojob = Output2.gens(k).gojob
  idex3to.goop = Output2.gens(k).goop
  While ((idex2.gojob <> 0) And (idex3.goop <> 0))
     j = j + 1
     idex2.gojob = Output1.gens(j).gojob
     idex3.goop = Output1.gens(j).goop
  Wend
  While ((idex2to.gojob <> 0) And (idex3to.goop <> 0))
     k = k + 1
     idex2to.gojob = Output2.gens(k).gojob
     idex3to.goop = Output2.gens(k).goop
```

```
        Wend
        Output1.gens(j).gojob = idx2.gojob
        Output1.gens(j).goop = idx3.goop
        Output2.gens(k).gojob = idx2to.gojob
        Output2.gens(k).goop = idx3to.goop
        For w = 1 To leng
         If p1.gens(w).gojob = idx2.gojob Then p1.gens(w).gojob = 0
         If p2.gens(w).gojob = idx2.gojob Then p2.gens(w).gojob = 0
         If p3.gens(w).gojob = idx2to.gojob Then p3.gens(w).gojob = 0
         If p4.gens(w).gojob = idx2to.gojob Then p4.gens(w).gojob = 0
         If p1.gens(w).goop = idx3.goop Then p1.gens(w).goop = 0
         If p2.gens(w).goop = idx3.goop Then p2.gens(w).goop = 0
         If p3.gens(w).goop = idx3to.goop Then p3.gens(w).goop = 0
         If p4.gens(w).goop = idx3to.goop Then p4.gens(w).goop = 0
        Next

      End Select
    Next
  End Select
```

Private Function mutation **untuk melakukan proses mutasi pada** ***job based representation. Function*** **ini terdapat dalam** ***class CGenetic***

```
i1 = Int(Rnd * JumlahJob + 1)
Do
    i2 = Int(Rnd * JumlahJob + 1)
Loop While i1 = i2

Select Case MutMethod

  Case Reciprocal
     Temp = mutation.Job(i1)
     mutation.Job(i1) = mutation.Job(i2)
     mutation.Job(i2) = Temp

  Case Insertion
     If i1 < i2 Then
        Temp = mutation.Job(i2)
        For i = i2 To i1 + 1 Step -1
           mutation.Job(i) = mutation.Job(i - 1)
        Next
        mutation.Job(i1) = Temp
     End If
     If i1 > i2 Then
        Temp = mutation.Job(i2)
```

```
        For i = i2 To i1 - 2
          mutation.Job(i) = mutation.Job(i + 1)
        Next
        mutation.Job(i1 - 1) = Temp
      End If

    End Select
```

Private Function mutation pada operation based representation. Function ini terdapat dalam class CGenetic2

```
Tempjob = k.gens(i1).gojob
Tempopr = k.gens(i1).goop

For i = i1 - 1 To 1 Step -1
  comjob(i) = k.gens(i).gojob
  If comjob(i) = Tempjob Then Exit For
  If ((i = 1) And (comjob(i) <> Tempjob)) Then cjob = False
Next

For j = i1 + 1 To leng
  comjob(j) = k.gens(j).gojob
  If comjob(j) = Tempjob Then Exit For
  If ((j = leng) And (comjob(j) <> Tempjob)) Then copr = False
Next

rentang = j - i - 1
If cjob = False Then
  rentang = rentang + 1
  bawah = 1
Else
  bawah = i
End If

If copr = False Then
  rentang = rentang + 1
  atas = leng
Else
  atas = j
End If

Do
  i2 = Int(Rnd * rentang + 1)
Loop While i2 = i1
```

```
'gen exchange
tampunggen = k.gens(i1)
k.gens(i1) = k.gens(i2)
k.gens(i2) = tampunggen
```

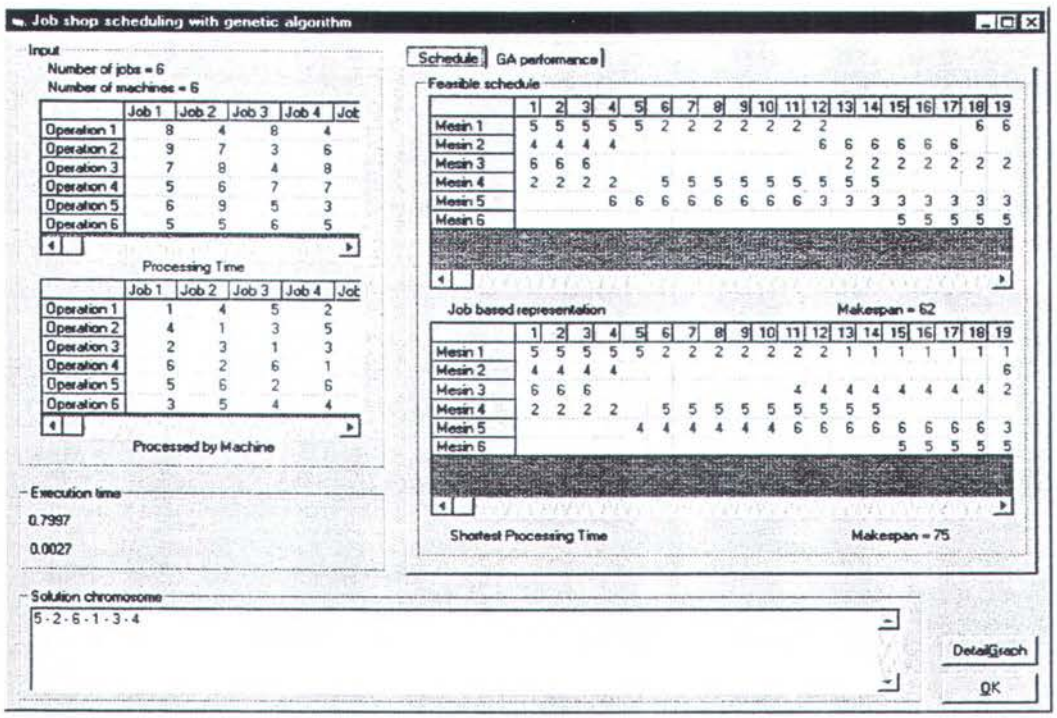
Private Sub RouletteWheel terdapat pada class CGenetic dan CGenetic2

```
'Menghitung total fitness value
FitTotal = 0
For i = 1 To PopSize
  FitTotal = FitTotal + dtPopulasi(i).Fitness
Next

'Menghitung probabilitas seleksi setiap kromosom
For i = 1 To PopSize
  dtPopulasi(i).RFitness = dtPopulasi(i).Fitness / FitTotal
  dtPopulasi(i).CFitness = dtPopulasi(i).RFitness
  If i > 1 Then dtPopulasi(i).CFitness = dtPopulasi(i).CFitness +
dtPopulasi(i - 1).CFitness
Next

'Seleksi
ReDim TmpPop(PopSize)
For idx = 1 To PopSize
  RndNum = Rnd
  If RndNum <= dtPopulasi(1).CFitness Then
    TmpPop(idx) = dtPopulasi(1)
  Else
    For i = 1 To PopSize - 1
      If (RndNum > dtPopulasi(i).CFitness) And _
        (RndNum <= dtPopulasi(i + 1).CFitness) Then
          TmpPop(idx) = dtPopulasi(i + 1)
      End If
    Next i
  End If
Next idx

ReDim dtPopulasi(PopSize)
For idx = 1 To PopSize
  dtPopulasi(idx) = TmpPop(idx)
Next
```

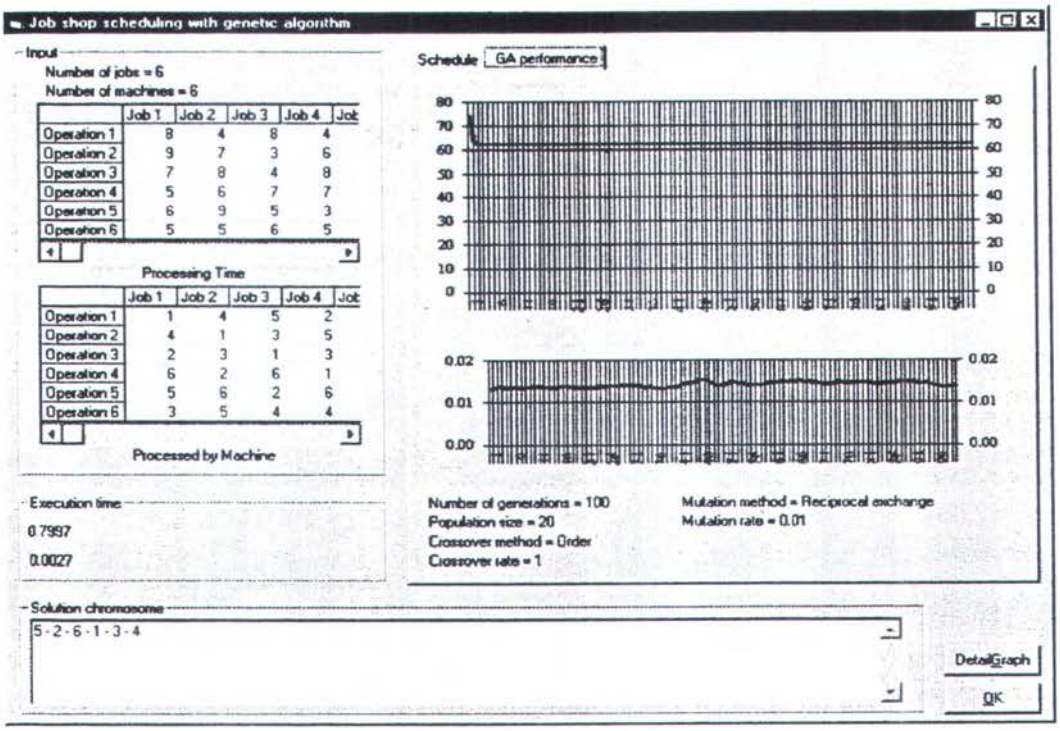
V.2.3. Tampilan keluaran

Berikut ini adalah tampilan keluaran yang terdiri atas :

- Detil data input
- Jadual, *makespan*, dan kromosom terbaik
- Grafik rata – rata nilai *fitness* setiap generasi
- Waktu eksekusi program



Gambar 5.6 Tampilan keluaran jadual



Gambar 5.7 Tampilan keluaran grafik rata – rata nilai *fitness*

BAB VI

UJI COBA DAN ANALISA

Pada kenyataannya tidak ada solusi optimal untuk penjadualan *n x m*. Dan tidak ada parameter yang baku untuk menguji kinerja algoritma genetik untuk memecahkan permasalahan penjadualan *job shop*.

Permasalahan yang digunakan untuk uji coba adalah 6x6, 10x10, 15x15, 5x16, 5x28, 18x5, dan 20x5. Data *job* dan mesin yang digunakan adalah data yang diberikan sendiri dan penjadualan *job shop* yang digunakan adalah *job shop* klasik dengan pola kedatangan *job* statis.

Data *job* dan mesin yang digunakan adalah sebagai berikut :

- Dimensi permasalahan 6x6

Tabel 6.1
Data *job* permasalahan 6x6

Operasi	Waktu pengerjaan (dalam satuan waktu)					
	Job					
	1	2	3	4	5	6
1	8	4	8	4	5	3
2	9	7	3	6	9	8
3	7	8	4	8	7	6
4	5	6	7	7	3	5
5	6	9	5	3	4	7
6	3	5	6	5	6	4

Tabel 6.2
Data mesin permasalahan 6x6

Operasi	Mesin pemroses					
	Job					
	1	2	3	4	5	6
1	1	4	5	2	1	3
2	4	1	3	5	4	5
3	2	3	1	3	6	2
4	6	2	6	1	5	1
5	5	6	2	6	3	4
6	3	5	4	4	2	6

- Dimensi permasalahan 10x10

Tabel 6.3
Data *job* permasalahan 10x10

Operasi	Waktu pengerjaan (dalam satuan waktu)									
	Job									
	1	2	3	4	5	6	7	8	9	10
1	3	10	4	4	11	5	8	2	3	2
2	5	7	2	6	2	7	5	4	6	6
3	4	5	6	3	8	3	2	6	4	4
4	5	11	6	7	4	7	8	9	5	7
5	8	4	5	7	6	3	9	12	8	10
6	7	5	8	5	12	5	11	6	6	3
7	6	14	7	11	4	3	10	3	2	5
8	9	7	10	14	7	8	3	5	5	11
9	10	8	3	5	2	5	7	10	6	3
10	5	4	5	4	6	3	10	5	3	8

Tabel 6.4
Data mesin permasalahan 10x10

Operasi	Mesin pemroses									
	Job									
	1	2	3	4	5	6	7	8	9	10
1	1	4	5	2	7	3	6	2	6	5
2	4	7	3	5	4	5	8	4	9	8
3	2	3	7	3	8	2	5	8	7	4
4	6	8	6	10	5	7	9	1	10	7
5	5	6	2	6	9	4	2	7	4	1
6	8	9	4	4	2	10	4	3	2	6
7	10	5	1	8	10	6	1	10	5	2
8	9	2	9	1	3	8	3	5	3	10
9	3	10	8	9	1	1	10	6	1	3
10	7	1	10	7	6	9	7	9	8	9

- Dimensi permasalahan 15x15

Tabel 6.5
Data *job* permasalahan 15x15

Operasi	Waktu pengerjaan (dalam satuan waktu)														
	Job														
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	7	8	4	9	4	11	5	8	2	3	3	8	6	6	4
2	4	5	3	3	8	2	7	5	4	6	9	6	9	15	14
3	6	3	8	6	6	8	3	2	6	4	7	4	6	2	8
4	5	9	6	9	15	4	7	8	9	5	6	4	2	3	6
5	9	7	4	6	2	6	3	9	12	8	8	6	3	8	2
6	8	10	9	8	10	12	5	11	6	6	9	7	11	8	5
7	3	12	7	5	4	4	3	10	3	2	5	12	5	6	9
8	10	5	16	2	6	4	2	3	2	3	7	11	3	9	7
9	4	14	8	7	8	6	3	8	4	8	5	7	4	6	4
10	12	7	5	6	4	5	9	6	5	6	9	5	8	2	13
11	8	4	9	8	6	8	5	4	6	7	4	10	12	5	12
12	5	8	4	5	5	3	8	3	10	5	8	4	4	3	6
13	14	6	7	8	9	6	8	3	8	10	12	6	4	2	5
14	2	9	5	6	7	15	4	7	5	4	4	3	5	7	10
15	8	9	8	4	5	2	6	3	2	6	4	2	7	2	8

Tabel 6.6
Data *job* permasalahan 15x15

Operasi	Mesin pemroses														
	Job														
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	1	7	12	9	3	5	2	11	15	13	8	10	4	6	14
2	4	5	7	5	10	3	14	6	5	3	2	5	1	13	6
3	15	4	9	8	2	15	11	3	8	11	7	6	14	2	7
4	3	14	4	2	8	4	4	7	12	7	5	3	12	8	11
5	5	12	1	1	7	13	10	9	7	9	14	11	6	4	9
6	7	6	5	13	13	2	1	12	4	4	3	13	3	3	10
7	2	2	8	14	9	9	3	4	6	1	15	2	11	9	8
8	9	15	3	12	6	11	6	15	11	5	4	4	7	10	12
9	14	1	11	6	12	14	12	13	2	14	13	7	5	5	4
10	11	8	13	3	5	10	5	5	9	8	1	15	13	12	1
11	10	11	2	11	1	6	7	1	3	2	6	1	9	1	2
12	8	3	15	7	14	1	13	10	10	15	10	9	10	14	15
13	13	9	14	10	11	7	8	2	14	10	12	14	2	7	13
14	6	10	6	15	4	12	15	8	13	6	9	8	15	11	5
15	12	13	10	4	15	8	9	14	1	12	11	12	8	15	3

- Dimensi permasalahan 5x16

Tabel 6.7
Data *job* permasalahan 5x16

Operasi	Waktu pengerjaan (dalam satuan waktu)				
	Job				
	1	2	3	4	5
1	3	8	2	7	5
2	6	6	8	3	2
3	9	15	4	7	8
4	6	2	6	3	9
5	8	10	12	5	11
6	5	4	4	3	10
7	2	6	4	2	3
8	3	10	5	8	4
9	3	8	10	12	6
10	7	5	4	4	3
11	8	4	5	5	3
12	6	7	8	9	6
13	9	5	6	7	15
14	9	8	4	5	2
15	5	4	6	9	6
16	2	6	4	7	4

Tabel 6.8
Data mesin permasalahan 5x16

Operasi	Mesin pemroses				
	Job				
	1	2	3	4	5
1	3	6	5	13	3
2	7	2	8	5	6
3	12	16	3	12	8
4	1	4	15	7	14
5	5	12	7	9	15
6	16	10	4	15	7
7	14	5	9	2	10
8	2	9	10	6	1
9	10	14	13	10	4
10	6	11	16	14	9
11	9	15	12	11	2
12	4	7	11	16	12
13	11	13	6	8	16
14	8	3	2	1	11
15	13	1	14	3	13
16	15	8	1	4	5

- Dimensi permasalahan 5x28

Tabel 6.9
Data *job* permasalahan 5x28

Operasi	Waktu pengerjaan (dalam satuan waktu)				
	Job				
	1	2	3	4	5
1	2	6	4	2	3
2	3	10	5	8	4
3	3	8	10	12	6
4	7	5	4	4	3
5	2	3	2	3	7
6	3	8	4	8	5
7	4	11	4	3	4
8	6	2	3	7	6
9	10	6	7	5	5
10	7	4	10	12	5
11	5	8	4	4	3
12	10	12	6	4	2
13	3	12	7	5	4
14	10	5	16	2	6
15	4	14	8	7	8
16	12	7	5	6	4
17	8	4	9	8	6
18	5	8	4	5	5
19	14	6	7	8	9
20	4	8	4	5	3
21	7	3	6	9	8
22	8	4	8	7	6
23	6	7	7	3	5
24	9	5	3	4	7
25	5	6	5	6	4
26	8	10	12	5	11
27	5	4	4	3	10
28	8	6	5	9	12

Tabel 6.10
Data mesin permasalahan 5x28

Operasi	Mesin pemroses				
	Job				
	1	2	3	4	5
1	1	7	2	9	3
2	5	3	8	6	18
3	7	14	21	25	5
4	3	2	3	5	6
5	24	8	23	2	26
6	6	16	18	3	9
7	28	9	12	7	16
8	4	5	4	8	10
9	16	1	17	24	4
10	23	12	5	21	2
11	18	24	26	13	8
12	2	27	19	18	17
13	12	11	10	20	7
14	22	6	13	14	21
15	8	19	6	22	11
16	9	23	16	12	14
17	14	20	7	17	24
18	13	10	11	23	13
19	21	18	24	26	25
20	10	21	27	15	1
21	20	15	9	4	27
22	26	22	25	27	12
23	15	26	14	10	15
24	19	4	28	11	20
25	27	25	1	28	19
26	25	28	15	1	23
27	17	13	20	16	28
28	11	17	22	19	22

- Dimensi permasalahan 18x5

Tabel 6.11
Data *job* permasalahan 18x5

Operasi	Waktu pengerjaan (dalam satuan waktu)																	
	Job																	
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
1	3	4	2	3	11	4	3	4	2	3	3	4	7	6	6	8	10	5
2	7	6	3	8	2	3	7	6	3	8	5	6	4	9	15	4	7	9
3	5	5	9	6	6	7	5	5	9	6	12	5	11	6	2	6	3	4
4	7	8	5	4	3	12	7	8	5	4	4	3	10	8	10	12	5	6
5	10	3	8	3	12	6	5	7	14	12	5	9	12	5	4	7	9	2

Tabel 6.12
Data mesin permasalahan 18x5

Operasi	Mesin pemroses																	
	Job																	
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
1	1	4	2	3	1	4	5	4	2	3	1	4	5	2	5	3	4	5
2	4	2	3	1	2	3	4	2	3	5	4	3	4	3	4	4	1	2
3	2	5	1	2	5	1	1	5	4	2	2	5	2	1	1	2	3	4
4	5	1	5	4	3	5	3	1	5	4	3	1	3	4	3	1	5	1
5	3	3	4	5	4	2	2	3	1	1	5	2	1	5	2	5	2	3

- Dimensi permasalahan 20x5

Tabel 6.13
Data *job* permasalahan 20x5

Operasi	Waktu pengerjaan (dalam satuan waktu)																			
	Job																			
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
1	4	6	8	4	5	6	5	2	4	2	4	11	4	3	4	2	3	2	3	7
2	2	7	3	6	9	8	7	6	5	5	6	2	3	7	6	3	8	4	8	5
3	6	4	6	5	6	6	9	4	2	4	10	6	7	5	5	9	6	5	6	9
4	5	6	4	7	2	5	2	7	8	6	3	3	12	7	8	5	4	6	7	4
5	8	3	2	4	5	7	4	6	5	2	7	6	5	10	3	8	3	10	5	8

Tabel 6.14
Data mesin permasalahan 20x5

Operasi	Mesin pemroses																			
	Job																			
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
1	1	5	4	2	3	4	3	4	2	3	3	1	2	4	5	3	2	2	1	4
2	4	3	3	5	4	5	2	3	5	1	4	2	1	5	2	4	5	3	3	2
3	3	1	1	3	1	2	5	1	4	5	2	5	3	2	4	2	1	5	4	5
4	5	2	5	1	5	1	4	5	3	4	5	3	5	3	1	5	4	1	2	3
5	2	4	2	4	2	3	1	2	1	2	1	4	4	1	3	1	3	4	5	1

VI.1. Uji coba parameter algoritma genetik

Parameter yang diuji adalah ukuran populasi, jumlah generasi, metode dan tingkat tukar silang, metode dan tingkat mutasi.

VI.1.1. Mendapatkan parameter terbaik

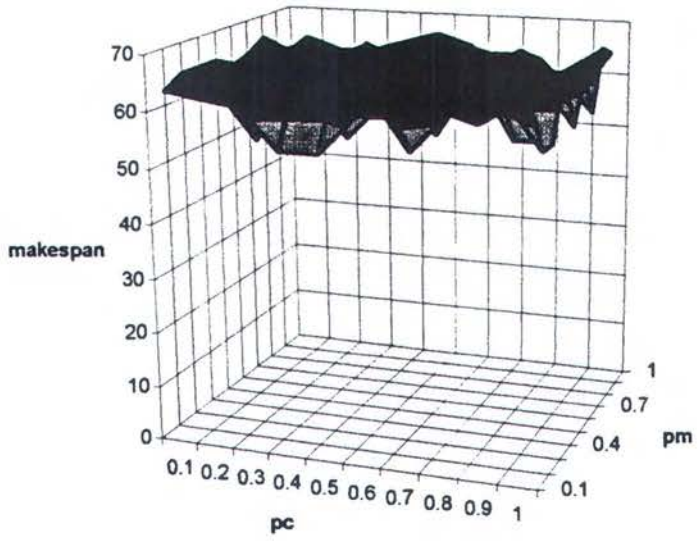
Agar didapatkan parameter terbaik untuk tingkat tukar silang dan tingkat mutasi dilakukan percobaan pada beberapa kombinasi parameter. Uji coba dilakukan pada dimensi permasalahan 6x6, 5x16, 5x28, dan 10x10.

- Dimensi permasalahan 6x6

Pada dimensi permasalahan 6x6, uji coba menggunakan ukuran populasi = 20 dan jumlah generasi = 100. Hasil uji coba adalah sebagai berikut :

Tabel 6.15
Uji coba parameter pada dimensi permasalahan 6x6

Makespan (dalam satuan waktu)		tingkat tukar silang									
		0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
tingkat mutasi	0.1	64	63	62	55	55	62	62	64	62	66
	0.2	66	63	55	66	64	64	55	62	64	64
	0.3	66	64	64	64	55	62	66	66	64	55
	0.4	66	65	55	66	66	62	55	66	55	55
	0.5	64	64	64	66	66	62	64	64	65	66
	0.6	65	62	64	62	64	55	62	64	64	55
	0.7	67	64	55	67	64	63	64	62	63	63
	0.8	64	64	64	62	63	55	62	66	62	55
	0.9	63	62	62	55	63	55	64	64	55	66
	1	64	62	62	64	66	64	55	62	55	64



Gambar 6.1 Uji coba parameter pada dimensi permasalahan 6x6

Dengan melihat hasil uji coba didapatkan bahwa untuk dimensi permasalahan 6x6 dengan ukuran populasi = 20 dan jumlah generasi = 100 dapat mencapai *makespan* minimal dengan menggunakan kombinasi Pc dan Pm sebagai berikut :

- Pc = 1 dengan Pm = 0.3 dan 0.4
- Pc = 0.9 dengan Pm = 0.4
- Pc = 0.7 dengan Pm = 0.3 dan 0.4

- Dimensi permasalahan 5x16

Pada dimensi permasalahan 5x16, uji coba menggunakan ukuran populasi = 10 dan jumlah generasi = 100. Hasil uji coba adalah sebagai berikut :

Tabel 6.16
Uji coba parameter pada dimensi permasalahan 5x16

Makespan (dalam satuan waktu)		tingkat tukar silang									
		0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
tingkat mutasi	0.1	120	120	120	120	120	120	120	120	120	120
	0.2	120	121	120	121	120	121	121	120	120	121
	0.3	120	120	120	121	120	121	120	121	120	120
	0.4	120	120	124	120	120	120	120	120	120	120
	0.5	124	121	121	120	120	120	120	120	121	120
	0.6	120	120	120	120	120	120	120	120	120	120
	0.7	120	120	120	121	120	120	120	121	120	120
	0.8	120	120	121	121	121	120	120	120	120	120
	0.9	120	124	120	120	120	120	120	121	120	120
	1	120	120	120	120	120	121	120	121	121	120

Hasil uji coba tidak menunjukkan pengaruh parameter algoritma genetik terhadap permasalahan 5x16. Hal ini disebabkan kromosom solusi menggunakan metode *job based representation*. Dengan metode ini penjadualan dipengaruhi oleh permutasi *job* dan kromosom terbaik telah dapat ditemukan dengan menggunakan parameter ukuran populasi = 10 dan jumlah generasi = 100.

- Dimensi permasalahan 5x28

Pada dimensi permasalahan 5x28, uji coba menggunakan ukuran populasi = 10 dan jumlah generasi = 100. Hasil uji coba adalah sebagai berikut :

Tabel 6.17
Uji coba parameter pada dimensi permasalahan 5x28

Makespan (dalam satuan waktu)		tingkat tukar silang									
		0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
tingkat mutasi	0.1	193	193	193	193	193	193	193	193	193	193
	0.2	193	193	193	193	193	193	193	193	193	193
	0.3	193	193	193	199	193	193	193	193	193	199
	0.4	193	193	199	193	193	193	193	193	193	193
	0.5	193	193	193	193	193	193	199	193	193	193
	0.6	193	193	193	193	193	193	193	193	199	193
	0.7	193	193	193	193	193	193	193	193	193	199
	0.8	193	193	193	193	193	193	193	193	193	193
	0.9	193	193	193	193	193	193	193	193	193	193
	1	193	193	193	193	193	193	193	193	199	193

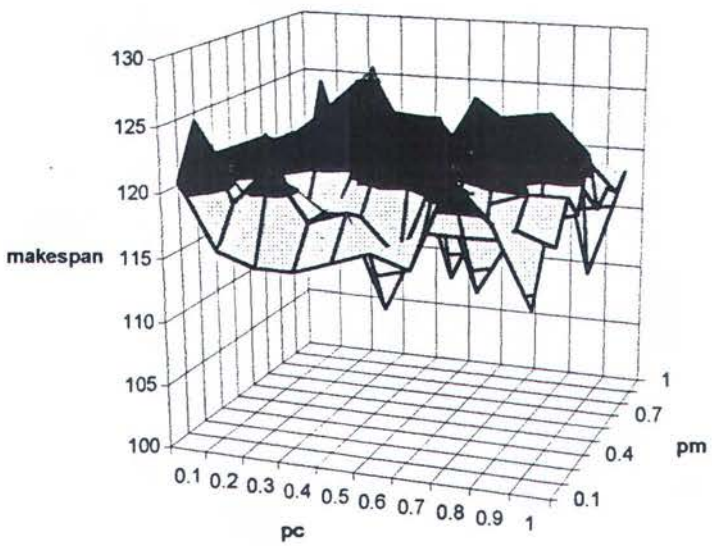
Seperti halnya permasalahan 5x16, bahwa dengan parameter ukuran populasi = 10 dan jumlah generasi = 100 telah memenuhi tujuan penjadualan, yaitu mencapai *makespan* minimal. Jika ukuran populasi diperbesar, maka peluang ditemukannya *makespan* minimal lebih besar.

- Dimensi permasalahan 10x10

Pada dimensi permasalahan 10x10, uji coba menggunakan ukuran populasi = 20 dan jumlah generasi = 100. Hasil uji coba adalah sebagai berikut :

Tabel 6.18
Uji coba parameter pada dimensi permasalahan 10x10

Makespan (dalam satuan waktu)		tingkat tukar silang									
		0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
tingkat mutasi	0.1	121	116	115	115	116	117	116	123	120	114
	0.2	125	119	122	118	119	112	118	118	118	118
	0.3	122	118	120	118	118	115	122	113	118	117
	0.4	119	123	116	118	121	123	113	119	120	120
	0.5	116	115	121	120	125	123	121	125	117	118
	0.6	121	120	120	122	124	124	116	116	125	122
	0.7	120	122	117	116	123	121	119	124	121	117
	0.8	118	123	120	123	118	116	115	121	118	118
	0.9	118	119	126	118	117	124	122	115	120	117
	1	124	115	112	118	114	117	120	122	109	118



Gambar 6.2 Uji coba parameter pada dimensi permasalahan 10x10

Dengan melihat hasil uji coba didapatkan bahwa untuk dimensi permasalahan 10x10 dengan ukuran populasi = 50 dan jumlah generasi = 500

dapat mencapai *makespan* minimal dengan menggunakan kombinasi Pc = 0.9 dan Pm = 1.

VI.1.2. Ukuran populasi

Uji coba ukuran populasi dan jumlah generasi dilakukan pada dimensi permasalahan 6x6, 5x16, 10x10, 18x5, dan 20x5 dengan menggunakan kombinasi Pc dan Pm yang telah dihasilkan sebelumnya. Uji coba ini bertujuan dapat ditentukan bahwa dengan ukuran populasi yang diperbesar dapat memperbaiki *makespan* yang dihasilkan.

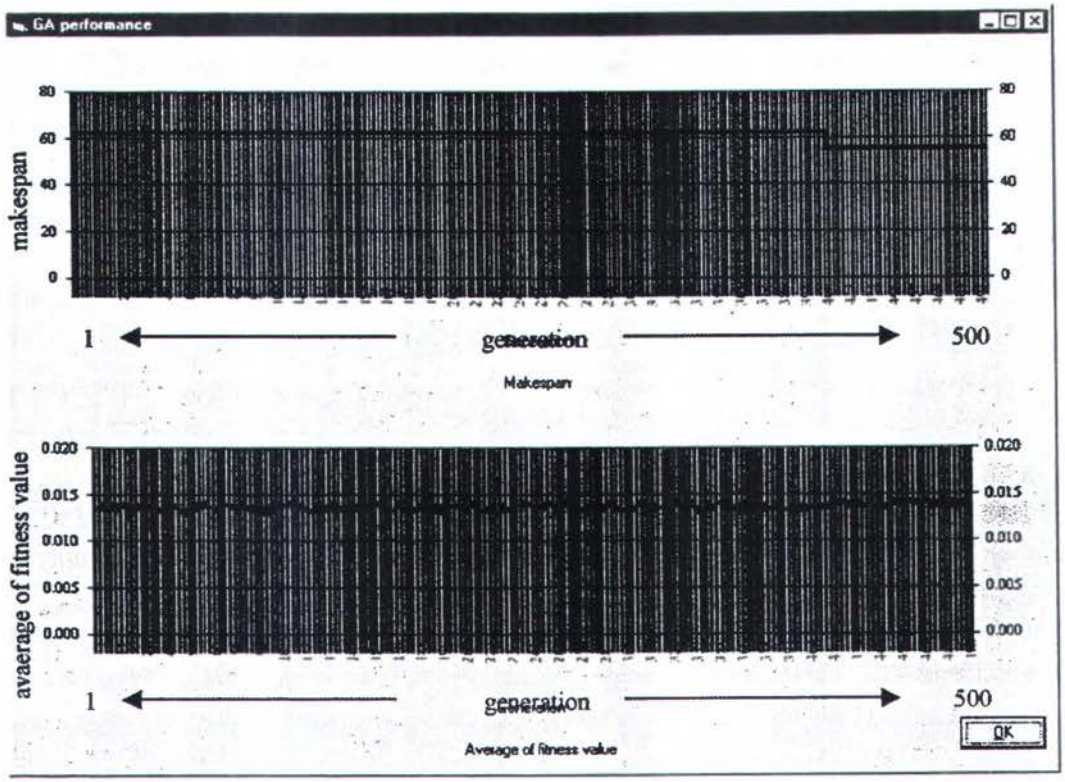
- Dimensi permasalahan 6x6

Uji coba pada dimensi permasalahan 6x6 menggunakan parameter sebagai berikut :

Tabel 6.19
Parameter uji coba ukuran populasi pada dimensi permasalahan 6x6

Parameter	Nilai
Dimensi permasalahan	6x6
Jumlah generasi	500
Ukuran populasi	20 dan 100
Metode tukar silang : *order*	1
Metode mutasi : *reciprocal exchange*	0.4

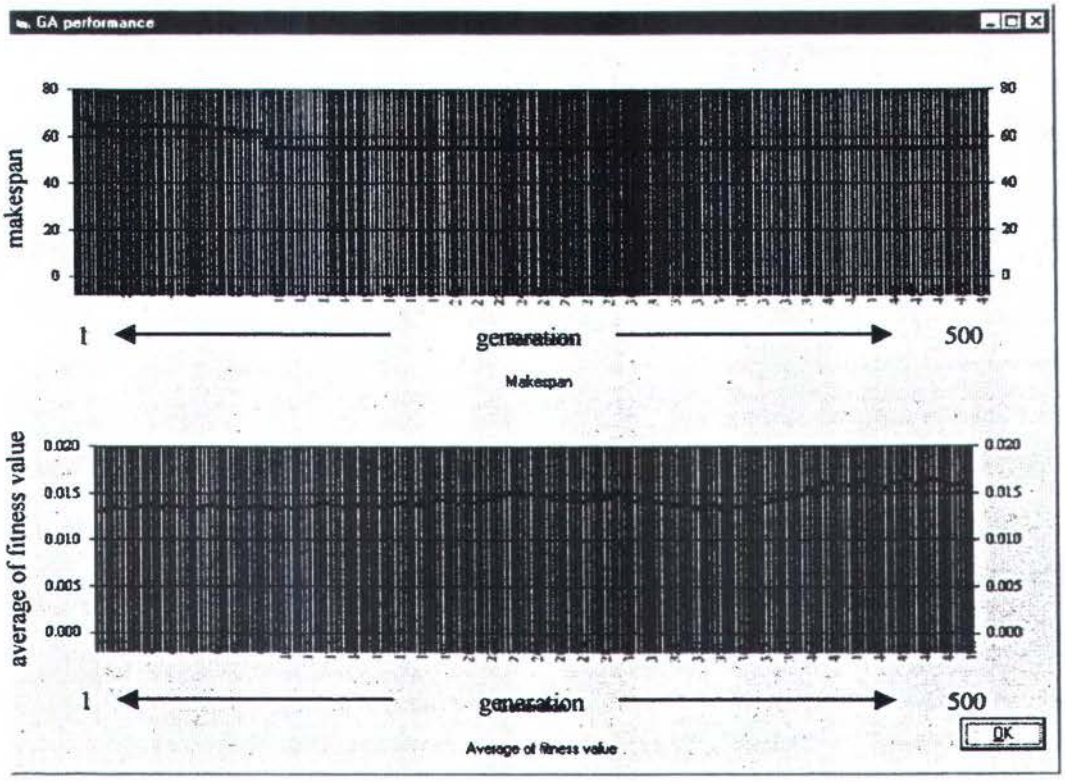
Hasil uji coba untuk ukuran populasi = 20 didapatkan sebagai berikut :



makespan = 55

Gambar 6.3 Hasil uji coba ukuran populasi = 20 pada dimensi permasalahan 6x6

Berikut ini adalah uji coba dengan ukuran populasi = 100. Hasil uji didapatkan sebagai berikut :



makespan = 55

Gambar 6.4 Hasil uji coba ukuran populasi = 100 pada dimensi permasalahan 6x6

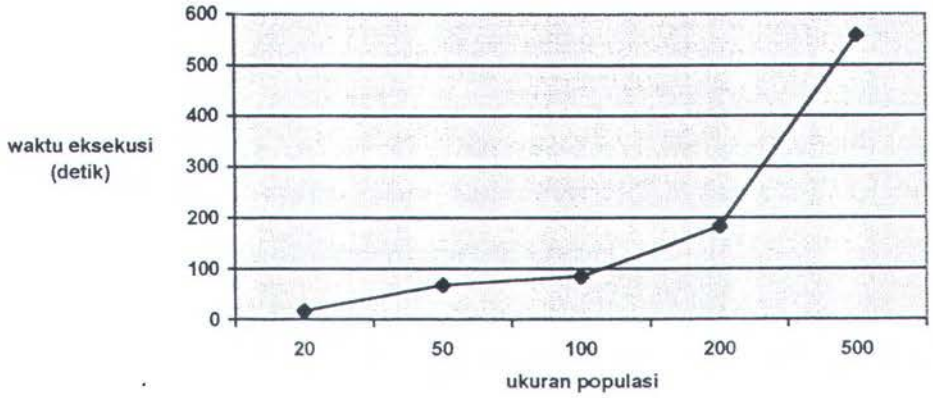
Hasil uji coba pada ukuran populasi = 100 menunjukkan bahwa pada grafik rata – rata nilai *fitness* dapat dilihat pembentukan populasi dari generasi ke generasi menunjukkan kemiripan populasi pada setiap generasi yang ditunjukkan oleh garis grafik yang lebih halus. Hal ini menunjukkan bahwa ukuran populasi yang diperbesar mengakibatkan varian kromosom yang lebih banyak, sehingga generasi berikutnya yang dihasilkan mirip dengan generasi sebelumnya. Ukuran populasi yang diperbesar juga mengakibatkan waktu eksekusi semakin besar.

Berikut ini adalah uji coba untuk mengamati hubungan antara ukuran populasi dengan waktu eksekusi :

Tabel 6.20
Pengaruh ukuran populasi terhadap waktu eksekusi pada dimensi permasalahan 6x6

Ukuran populasi	Jumlah generasi	Waktu eksekusi (detik)
20	100	17.088
50	100	68.234
100	100	85.890
200	100	182.310
500	100	558.755

Semakin besar ukuran populasi mengakibatkan waktu eksekusi semakin lama. Peningkatan waktu eksekusi ini dapat dilihat pada grafik berikut :



Gambar 6.5 Pengaruh ukuran populasi terhadap waktu eksekusi pada dimensi permasalahan 6x6

Jadi, untuk permasalahan 6x6, algoritma genetik dapat mencapai *makespan* minimal dengan parameter sebagai berikut :

- ukuran populasi = 20
- jumlah generasi = 100
- Pc = 1 dan Pm = 0.4.

Dengan parameter tersebut dibutuhkan waktu eksekusi selama 17.088 detik

- Dimensi permasalahan 5x16

Uji coba parameter ukuran populasi pada dimensi permasalahan 5x16 hanya ditujukan untuk mengetahui pengaruh ukuran populasi terhadap waktu eksekusi karena parameter terbaik untuk permasalahan 5x16 telah ditemukan. Parameter yang digunakan uji coba adalah sebagai berikut :

Tabel 6.21
Parameter uji coba ukuran populasi pada dimensi permasalahan 5x16

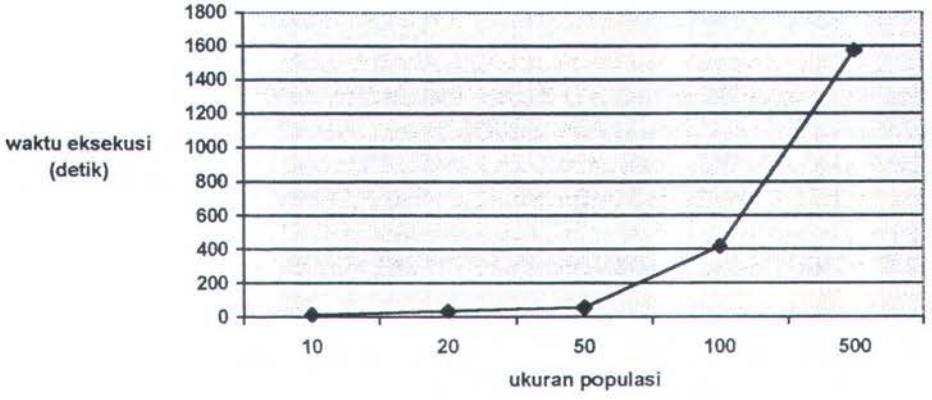
Parameter	Nilai
Dimensi permasalahan	5x16
Jumlah generasi	100
Ukuran populasi	10, 50, dan 100
Metode tukar silang : *order*	0.1
Metode mutasi : *reciprocal exchange*	0.1

Hasil uji coba adalah sebagai berikut :

Tabel 6.22
Pengaruh ukuran populasi terhadap waktu eksekusi pada dimensi permasalahan 5x16

Ukuran populasi	Jumlah generasi	Waktu eksekusi (detik)
10	100	13.780
20	100	34.559
50	100	55.446
100	100	416.119
500	100	1572.397

Semakin besar ukuran populasi mengakibatkan waktu eksekusi semakin lama. Peningkatan waktu eksekusi ini dapat dilihat pada grafik berikut :



Gambar 6.6 Pengaruh ukuran populasi terhadap waktu eksekusi pada dimensi permasalahan 5x16

Jadi, untuk permasalahan 5x16, algoritma genetik dapat mencapai *makespan* minimal dengan parameter sebagai berikut :

- ukuran populasi = 10
- jumlah generasi = 100
- Pc = 0.1 dan Pm = 0.1.

Dengan parameter tersebut dibutuhkan waktu eksekusi selama 13.780 detik.

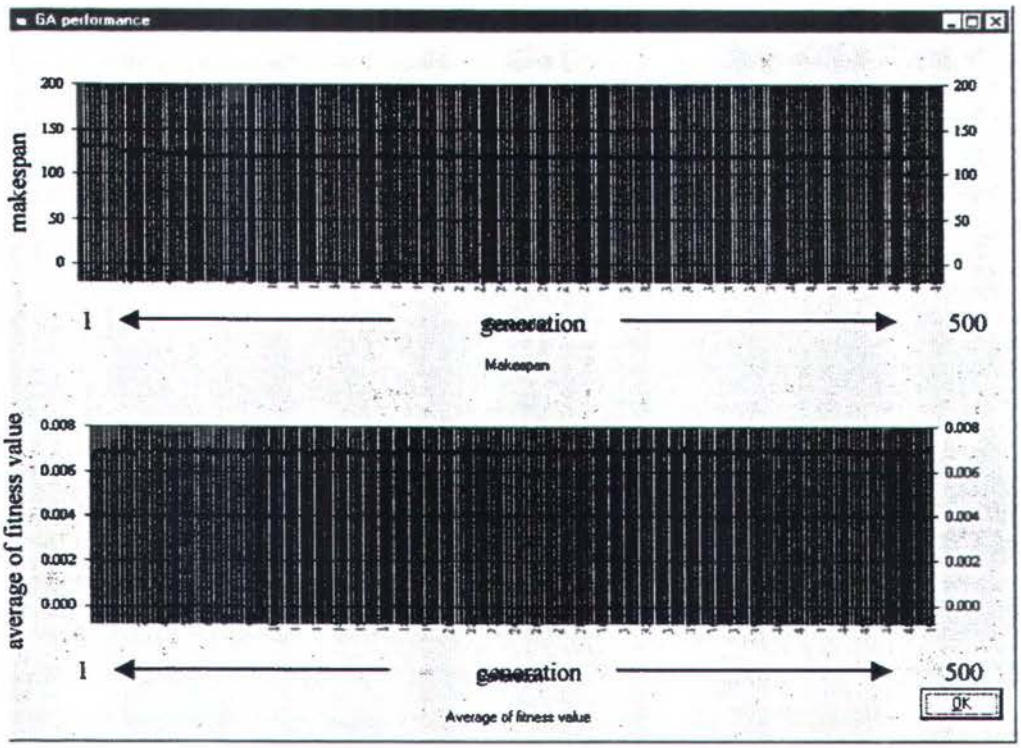
- Dimensi permasalahan 10x10

Uji coba parameter ukuran populasi pada dimensi permasalahan 10x10 untuk mendapatkan kemugkinan ditemukannya *makesapan* minimal yang lebih baik. Uji coba ini juga untuk mengetahui pengaruh ukuran populasi terhadap waktu eksekusi. Uji coba ini menggunakan parameter sebagai berikut :

Tabel 6.23
Parameter uji coba ukuran populasi pada dimensi permasalahan 10x10

Parameter	Nilai
Dimensi permasalahan	10x10
Jumlah generasi	500
Ukuran populasi	50 dan 500
Metode tukar silang : *order*	0.9
Metode mutasi : *reciprocal exchange*	1

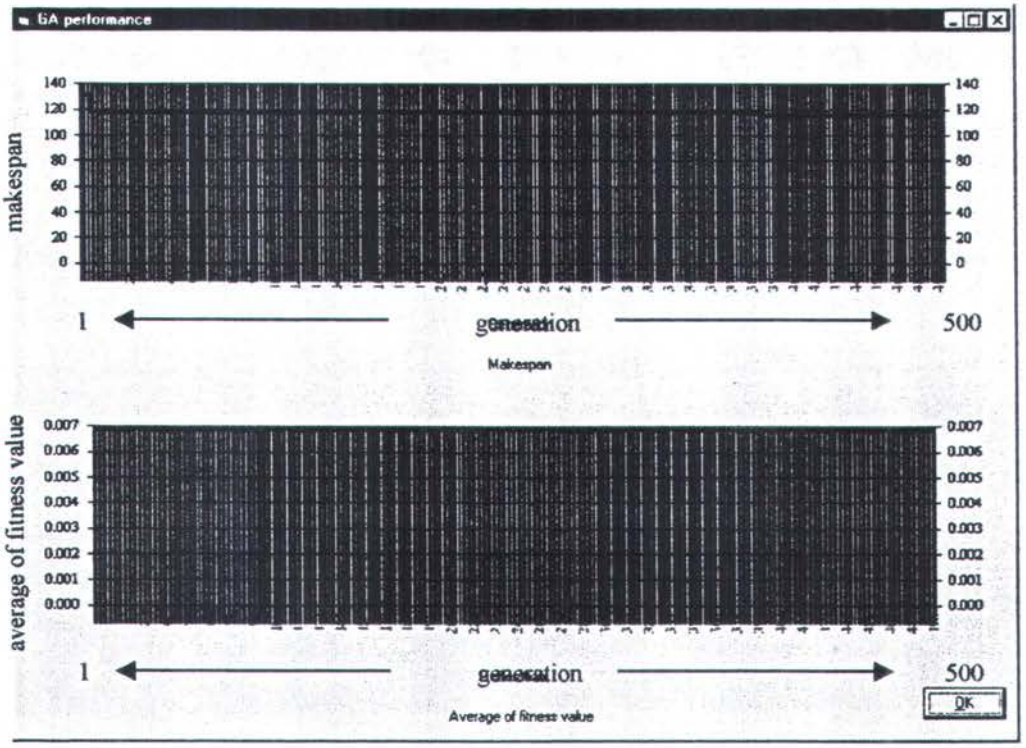
Hasil uji coba untuk ukuran populasi = 50 didapatkan sebagai berikut :



makespan=120

Gambar 6.7 Hasil uji coba ukuran populasi = 50 pada dimensi permasalahan 10x10

Berikut ini adalah uji coba dengan ukuran populasi = 500. Hasil uji didapatkan sebagai berikut :



makespan = 115

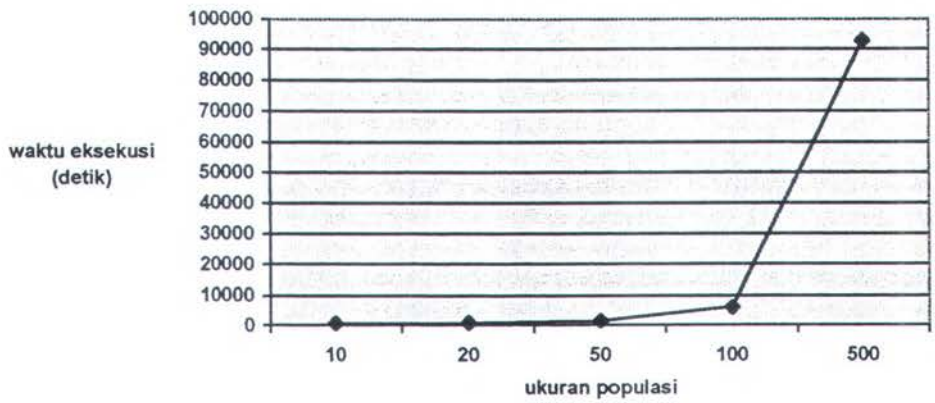
Gambar 6.8 Hasil uji coba ukuran populasi = 500 pada dimensi permasalahan 10x10

Hasil uji coba untuk mengamati pengaruh ukuran terhadap waktu eksekusi adalah sebagai berikut :

Tabel 6.24
Pengaruh ukuran populasi terhadap waktu eksekusi pada dimensi permasalahan 10x10

Ukuran populasi	Jumlah generasi	Waktu eksekusi (detik)
10	500	605.026
20	500	803.478
50	500	1494.500
100	500	5924.604
500	500	92939.788

Semakin besar ukuran populasi mengakibatkan waktu eksekusi semakin lama. Peningkatan waktu eksekusi ini dapat dilihat pada grafik berikut :



Gambar 6.9 Pengaruh ukuran populasi terhadap waktu eksekusi pada dimensi permasalahan 10x10

Jadi, untuk permasalahan 10x10, algoritma genetik dapat mencapai *makespan* minimal yang lebih baik dengan parameter sebagai berikut :

- ukuran populasi = 500
- jumlah generasi = 500
- Pc = 1 dan Pm = 0.9.

Dengan parameter tersebut dibutuhkan waktu eksekusi selama 92939.788 detik.

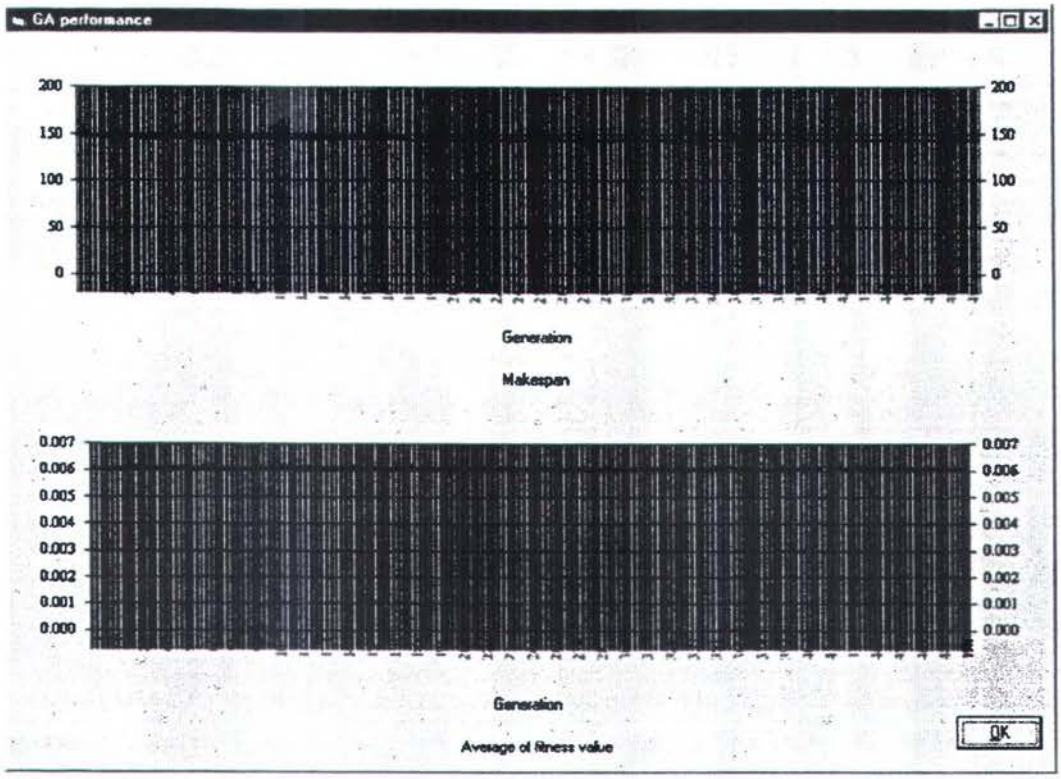
- Dimensi permasalahan 18x5

Uji coba parameter ukuran populasi pada dimensi permasalahan 18x5 hanya ditujukan untuk mengetahui pengaruh ukuran populasi terhadap waktu eksekusi. Uji coba ini menggunakan parameter sebagai berikut :

Tabel 6.25
Parameter uji coba ukuran populasi pada dimensi permasalahan 18x5

Parameter	Nilai
Dimensi permasalahan	18x5
Jumlah generasi	200
Ukuran populasi	100 dan 500
Metode tukar silang : *order*	0.6
Metode mutasi : *reciprocal exchange*	0.8

Hasil uji coba untuk ukuran populasi = 100 didapatkan sebagai berikut :



makespan=143

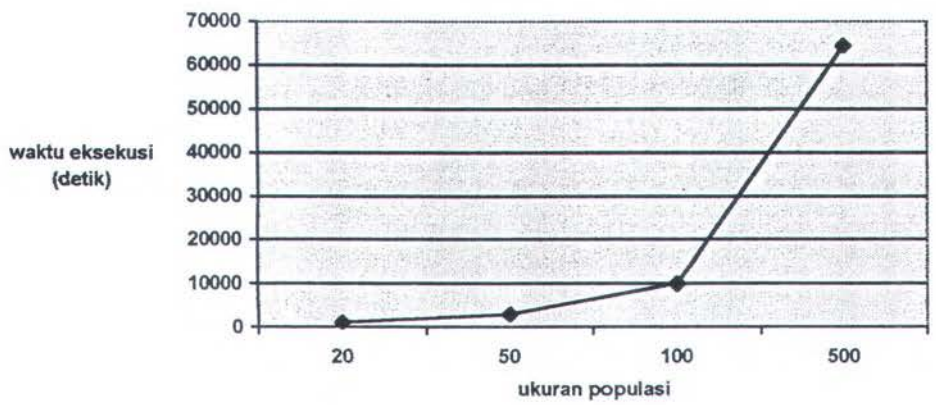
Gambar 6.10 Hasil uji coba ukuran populasi = 100 pada dimensi permasalahan 18x5

Hasil uji coba dengan memperbesar ukuran populasi adalah sebagai berikut :

Tabel 6.26
Pengaruh ukuran populasi terhadap waktu eksekusi pada dimensi permasalahan 18x5

Ukuran populasi	Jumlah generasi	Waktu eksekusi (detik)
20	200	1151.910
50	200	3033.346
100	200	9965.759
500	200	64392.471

Ukuran populasi yang diperbesar mempengaruhi waktu eksekusi yang semakin lama. Peningkatan waktu eksekusi tersebut dapat dilihat pada grafik berikut :



Gambar 6.11 Pengaruh ukuran populasi terhadap waktu eksekusi pada dimensi permasalahan 18x5

Jadi, untuk permasalahan 18x5, algoritma genetik dapat mencapai *makespan* minimal dengan parameter sebagai berikut :

- ukuran populasi = 500
- jumlah generasi = 200
- Pc = 0.6 dan Pm = 0.8.

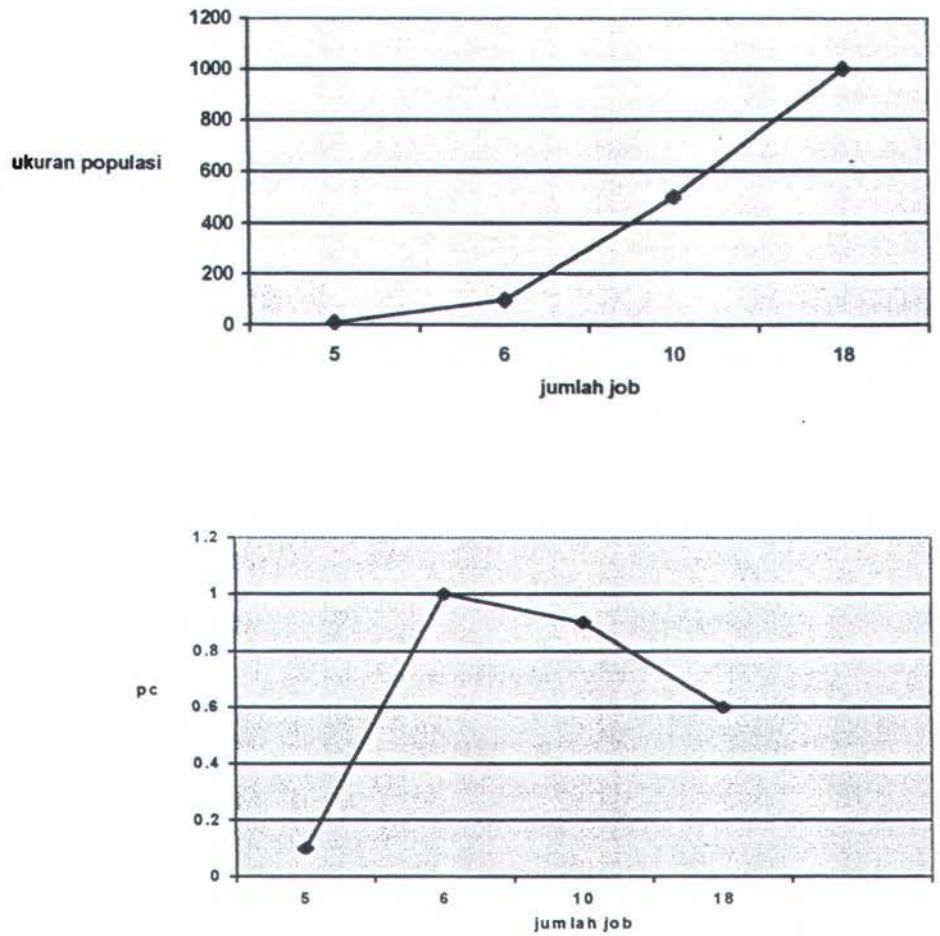
Dengan parameter tersebut dibutuhkan waktu eksekusi selama 64392.471 detik.

VI.1.3. Analisa parameter algoritma genetik

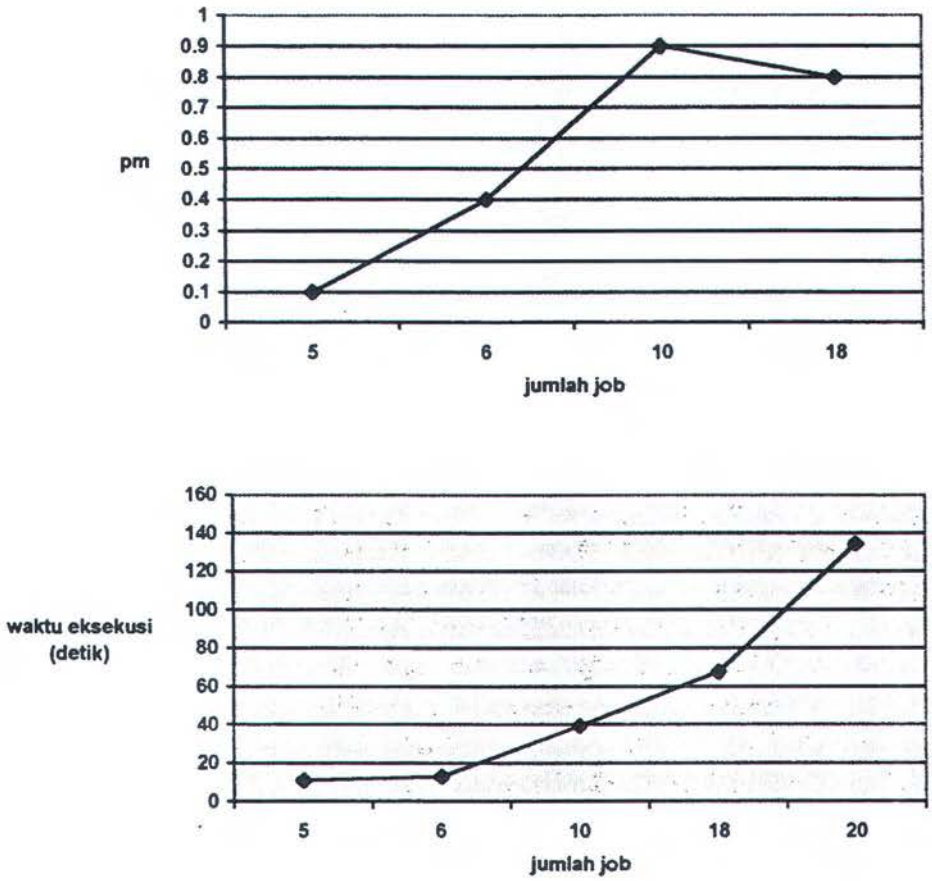
Dengan melihat hasil uji coba di atas diketahui pada masing – masing ukuran populasi, untuk mendapatkan individu terbaik, masing – masing diperoleh pada generasi yang bervariasi. Hal ini disebabkan adanya individu yang lebih banyak dalam suatu populasi akan berakibat pada eksplorasi yang lebih luas pada saat generasi awal, sehingga individu terbaik dapat ditemukan lebih awal. Namun faktor acak pada pembentukan populasi awal juga berperan dalam hal menghasilkan populasi yang baik atau buruk. Bila faktor acak ini menghasilkan populasi awal yang sudah mengarah pada populasi terbaik, maka dapat juga ditemukan individu terbaik pada generasi awal.

Ukuran populasi yang diperbesar berpengaruh dalam memberikan ruang eksplorasi yang lebih luas agar didapat kemungkinan untuk membentuk populasi yang lebih baik lagi. Sedangkan jumlah generasi memberikan kesempatan untuk perbaikan populasi dari generasi ke generasi berikutnya. Hal ini tentu saja dipengaruhi oleh proses tukar silang dan mutasi.

Berikut ini dapat digambarkan perubahan parameter algoritma genetik bila dihubungkan dengan perubahan jumlah job :



Gambar 6.12 Hubungan antara jumlah *job* dengan parameter algoritma genetik dan waktu eksekusi



Gambar 6.12 Hubungan antara jumlah *job* dengan parameter algoritma genetik dan waktu eksekusi (lanjutan)

VI.2. Perbandingan algoritma genetik dengan heuristik

Algoritma genetik yang dibuat pada tugas akhir ini dibandingkan dengan algoritma heuristik yang menggunakan aturan prioritas (*priority dispatching rule*), dalam hal ini *shortest processing time* (SPT). Perbandingan yang dilakukan adalah *makespan* yang dicapai dan waktu eksekusi.

- Dimensi pemasalahan 6x6

Tabel 6.27
Perbandingan *makespan* pada dimensi permasalahan 6x6

Uji coba	Makespan GA	Makespan SPT	Uji coba	Makespan GA	Makespan SPT
1	64	75	11	55	75
2	55	75	12	64	75
3	66	75	13	55	75
4	55	75	14	64	75
5	55	75	15	62	75
6	55	75	16	64	75
7	63	75	17	55	75
8	55	75	18	55	75
9	55	75	19	55	75
10	62	75	20	64	75

- Dimensi permasalahan 5x16

Tabel 6.28
Perbandingan *makespan* pada dimensi permasalahan 5x16

Uji coba	Makespan GA	Makespan SPT	Uji coba	Makespan GA	Makespan SPT
1	120	136	11	120	136
2	120	136	12	120	136
3	120	136	13	120	136
4	120	136	14	120	136
5	120	136	15	120	136
6	120	136	16	120	136
7	120	136	17	120	136
8	120	136	18	120	136
9	120	136	19	120	136
10	120	136	20	120	136

- Dimensi permasalahan 20x5

Tabel 6.29
Perbandingan *makespan* pada dimensi permasalahan 20x5

Uji coba	Makespan GA	Makespan SPT	Uji coba	Makespan GA	Makespan SPT
1	129	171	11	133	171
2	130	171	12	132	171
3	132	171	13	132	171
4	129	171	14	133	171
5	132	171	15	125	171
6	133	171	16	130	171
7	132	171	17	133	171
8	130	171	18	129	171
9	133	171	19	132	171
10	131	171	20	131	171

VI.3. Perbandingan metode representasi

Ada beberapa metode representasi yang sudah diperkenalkan para peneliti dengan diikuti modifikasi oleh peneliti – peneliti selanjutnya. Namun demikian sangat sulit menentukan metode yang terbaik. Hal ini disebabkan metode tersebut diujicobakan dengan kondisi berbeda walaupun dibandingkan pada suatu *benchmark* yang sama. Kondisi tersebut adalah :

- Jumlah total kromosom yang telah dilibatkan dalam penelitian

  Karena algoritma genetik bekerja dengan kromosom yang jumlahnya dapat ditentukan dan random, maka terpilihnya kromosom – kromosom untuk dievaluasi sangat menentukan kinerja algoritma genetik yang implikasinya pada penyelesaian penjadualan *job shop* itu sendiri. Parameter algoritma

genetik yang digunakan masing – masing peneliti juga besar pengaruhnya pada efektivitas dan efisiensi kinerja algoritma genetik

- Jumlah uji coba yang dilakukan

  Pada dasarnya setiap uji coba adalah menghasilkan sesuatu yang random, sehingga frekuensi uji coba juga menentukan hasil penelitian, tetapi sifatnya masih sangat bervariasi karena sifat random algoritma genetik itu sendiri

- Kondisi komputasi tempat uji coba

  Uji coba yang dilakukan dengan spesifikasi komputasi yang berbeda akan mempengaruhi hasil penelitian, setidaknya mempengaruhi tingkat presisi dalam penghitungan parameter algoritma genetik.

Namun demikian beberapa metode representasi tersebut memiliki beberapa faktor kesamaan aspek komputasi, yaitu :

- Properti Lamark
- Kompleksitas pengkodean
- Teknik pengkodean (*coding dan mapping*)
- Teknik menyimpan dalam *memory*

VI.3.1. Properti Lamark

Properti Lamark menunjukkan kaitan langsung susunan gen antara *parent* dan *offspring* yang dihasilkan (gen yang diturunkan). Hal ini berkaitan erat dengan metode representasi yang digunakan sebab metode representasi sangat menentukan bagaimana menyusun gen dalam kromosom. Sehingga pada proses berikutnya melalui tukar silang dan mutasi dapat diketahui pasti gen – gen yang

akan dioperasikan. Metode representasi yang berbeda akan berbeda juga di dalam memperlakukan gen di dalam kromosom karena nilai yang dimiliki oleh gen pada suatu kromosom belum tentu menunjukkan gen yang sama dengan representasi yang lain.

Contoh sederhana adalah *random key representation*. Setelah dibangkitkan bilangan acak sejumlah gen, maka bilangan acak tersebut dipandang sebagai permutasi *job* atau operasi menurut besar kecilnya nilai bilangan acak tersebut. Pada kromosom lain tentunya nilai acak ini berbeda yang dibangkitkan, tetapi jika dipandang menurut besar kecilnya akan menimbulkan arti yang sama pada permutasi *job* atau operasi. Demikian pula dengan *operation based representation*. Jika suatu gen mengandung nilai 1 bermakna gen tersebut adalah *job* 1, tetapi belum menunjukkan operasi. Jadi masih dibutuhkan bantuan aspek pemrograman yang lain untuk mengidentifikasikan *job* dan operasi secara jelas. Namun berbeda dengan *random key representation*, pada *operation based representation*, sebagian makna dari nilai gen telah didapatkan. Jadi *random key representation* dikatakan tidak memiliki properti Lamark dan *operation based representation* dikatakan memiliki setengah dari properti Lamark. *Job based representation* dikatakan memiliki properti Lamark karena setiap gen mengandung nilai pasti menunjukkan *job* tertentu. Dan *job* itu pula yang dikenakan operasi genetik melalui tukar silang dan mutasi.

VI.3.2. Kompleksitas pengkodean

Pada dasarnya terdapat banyak jadual yang *feasible* yang dapat dibentuk. Dan pada umumnya, ada tiga macam jenis penjadualan, yaitu : semiaktif, aktif, dan *nondelay*. Jadual yang optimal dapat ditemukan pada himpunan dari jadual aktif. Jadual *nondelay* pada umumnya adalah lebih kecil daripada jadwal yang aktif, tapi tidak menjamin akan terdapat jadual optimal. Sehingga representasi kromosom solusi dalam algoritma genetik diharapkan dapat menuju pembentukan jadual aktif.

Tingkat perbandingan kompleksitas pengkodean (*complexity of decoder*) dari cara representasi kromosom dapat diklasifikasikan menjadi empat tingkatan dibawah ini :

Level 0 dapat dikatakan tidak ada pengkodean (*no decoder*) karena semua permasalahan direpresentasikan secara langsung pada operator genetik. Yang termasuk level ini adalah *completion time based representation*

Level 1 yaitu relasi *mapping* sederhana (*simple mapping relation*), maksudnya adalah mapping permasalahan ke dalam representasi kromosom. Yang dapat dikelompokkan pada level ini adalah *operation based representation*, *job pair relation based representation*, dan *job based representation.*

Level 2 yaitu heuristik sederhana (*simple heuristic*) . Yang termasuk dalam level ini adalah *preference list based representation* dan *priority rule based representation*

Level 3 adalah heuristik yang lebih kompleks (*complex heuristic*). Yang termasuk level ini adalah *disjunctive graph based representation*, dan *machine based representation*.

VI.3.3. Teknik pengkodean (coding dan mapping)

Pada setiap teknik pengkodean atau representasi, kromosom dan solusi selalu memiliki hubungan relasi pemetaan. Sehingga dapat dikatakan bahwa setiap metode representasi selalu menghasilkan kromosom – kromosom legal dan *feasible*. Dan jika solusi yang dihasilkan adalah jadual aktif, maka korespondensi tersebut berupa korespondensi 1-1.

Namun, pada kenyataan kemudian, seiring dengan proses dari algoritma genetik itu sendiri, beberapa representasi tersebut dapat menghasilkan kromosom – kromosom ilegal. Sehingga cara representasi ini dapat dikelompokkan menjadi dua, yaitu :

- Kelompok pertama adalah memuat representasi yang hanya menghasilkan kromosom – kromosom *feasible*. Yang termasuk dalam kelompok ini adalah *priority rule based representation*, *job based representation*, dan *disjunctive graph representation*.
- Kelompok kedua adalah representasi yang memungkinkan membentuk kromosom ilegal. Yang termasuk dalam kelompok ini adalah *completion time based representation*, *operation based representation*, *machine based representation*, *preference list based representation*, *job pair relation based representation*.

VI.3.4. Teknik penyimpanan kromosom

Pada permasalahan n *job* x m mesin, jika panjang kromosom standar adalah n x m, maka di dalam komputasi, setiap kromosom disimpan dalam bagian – bagian *memory* sepanjang n x m. Dan jika metode representasi yang telah dibuat dikelompokkan menurut ukuran standar tersebut, maka didapatkan tiga jenis metode representasi, yaitu :

- Metode representasi dengan menggunakan panjang kromosom yang lebih pendek dari standar. Ada beberapa metode yang termasuk di dalamnya, yaitu : *job based representation*, *disjunctive graph based representation*, dan *machine based representation.*
- Metode representasi dengan menggunakan panjang kromosom yang lebih panjang dari standar. Yang termasuk di dalamnya adalah : *job pair relation based representation.*
- Metode representasi dengan menggunakan panjang kromosom yang sama dengan standar. Yang termasuk dalam jenis ini adalah selain dari metode representasi yang telah disebutkan. *Operation based representation* termasuk di dalamnya.

Ukuran panjang kromosom ini secara komputasi mengakibatkan banyak atau sedikitnya area *memory* yang digunakan.

BAB VII

KESIMPULAN DAN SARAN

Bab ini menjelaskan kesimpulan yang dapat diambil dalam tugas akhir ini dan saran – saran untuk pengembangan di masa mendatang.

VII.1. Kesimpulan

Beberapa kesimpulan yang diperoleh setelah mengimplementasikan perangkat lunak dan melakukan uji coba terhadapnya adalah sebagai berikut :

1. Algoritma genetik dapat digunakan untuk menyelesaikan permasalahan penjadualan *job shop* dengan tujuan mencapai waktu terpendek untuk pengerjaan seluruh *job* yang disebut *makespan*
2. Dengan batasan permasalahan yang diberikan, dengan jumlah job < 10, maka parameter terbaik yang diperlukan adalah sebagai berikut :
   - Ukuran populasi = 500
   - Jumlah generasi = 200
   - Tingkat tukar silang (Pc) = 1
   - Tingkat mutasi (Pm) = 0.4
3. Untuk dimensi permasalahan selain yang telah diuji, parameter algoritma genetik yang dapat menghasilkan *makespan* terbaik adalah mengikuti grafik trend parameter algoritma genetik yang dijelaskan pada Bab VI
4. Algoritma genetik dapat menghasilkan *makespan* lebih kecil dari pada *shortest processing time* (SPT) dan waktu eksekusi juga lebih cepat.

VII.2. Saran – saran

Untuk mengembangkan tugas akhir ini, maka dibutuhkan beberapa hal yang berkaitan dengan pendekatan pada permasalahan dan aspek komputasi. Beberapa hal tersebut adalah :

1. Agar perangkat lunak yang dihasilkan lebih dekat pada penjadualan *job shop* sebenarnya, maka pada perangkat lunak ini dapat dimodifikasi dengan penambahan beberapa variabel penjadualan *job shop* seperti pada kenyataan yang terjadi pada sistem produksi
2. Untuk memperbaiki waktu eksekusi yang masih lebih lama dari metode heuristik, maka perlu disempurnakan pada algoritma genetik yang dibuat agar dapat dilakukan pemotongan generasi bila sudah tercapai konvergensi dini pada generasi tertentu dan tidak perlu memproses permasalahan sampai jumlah generasi terakhir
3. Perangkat keras yang digunakan seharusnya menggunakan komputer dengan spesifikasi komputasi untuk perhitungan kompleks, misalnya komputer *parallel* dan *pipeline*
4. Oleh karena pada implementasi algoritma genetik untuk penjadualan *job shop* dapat dilakukan dengan beberapa metode representasi kromosom selain kedua metode yang digunakan, maka perlu diadakan penelitian terhadap beberapa metode tersebut sebagai bahan pembanding

DAFTAR PUSTAKA

1. K.R. Baker, *Introduction to Sequencing and Scheduling*, John Wiley & Sons, Inc., New York, 1974

2. C. Bierwirth, D.C. Mattfeld, *Production Scheduling and Rescheduling with Genetic Algorithms, Journal of Evolutionary Computation*, Volume 7, Number 1, University of Bremen, 1995

3. L. Booker, *Improving search in genetic algorithms*, in Davis, L., editor, *Genetic Algorithms and Simulated Annealing*, Morgan Kauffman Publishers, Los Altos, CA, 1987

4. R. W. Conway, W. L. Maxwell, L. W. Miller, *Theory of Scheduling*, Addison-Wesley Publishing Company, Inc., New Jersey, 1967

5. E.A. Elsayed, T.O. Boucher, *Analysis and Control of Production Systems*, 2nd edition, Prentice Hall International, New Jersey, 1994, h.291

6. M. Gen, R. Cheng, *Genetic Algorithms and Engineering Design*, John Wiley & Sons, New York, 1996, h.191

7. D. E. Goldberg, *Genetic Algorithm in Search, Optimization, and Machine Learning*, Addison-Wesley, 1989

8. J.H. Holland, *Adaptation in Natural and Artificial Systems*, University of Michigan Press, Ann Arbor, 1975

9. Z. Michalewicz, *Genetic Algorithms + Data Structures = Evolutions Programs*, 3rd revised and extended edition, Springer, New York, 1996

10. T.E. Mortton, D.W. Pentico, *Heuristic Scheduling System with Application to Production Systems and Project Management*, John Wiley and Sons, Inc., Canada, 1993

11. M. Obitko, http://www.cs.felk.ut.cz , 1998

12. S. Sittisathanchai, C.H. Dagli, H.C.Lee, *A Genetic Scheduler for Job-Shops*, Department of Engineering Management, University of Missouri-Rolla, 1997

13. T. Yamada, R. Nakano, *Genetic Algorithms for Job-Shop Scheduling Problems*, Proceedings of Modern Heuristic for Decision Support, pp.67 – 81,UNICOM seminar, London, 1997