PENGARUH WAKTU PENAMBAHAN JOB BARU PADA PENJADWALAN JOB SHOP DINAMIS MENGGUNAKAN ALGORITMA GENETIK

SKRIPSI

Sebagai salah satu syarat untuk memperoleh gelar
Sarjana dalam bidang Ilmu Komputer

oleh
HASAN BASRI
0310960037-96

PROGRAM STUDI ILMU KOMPUTER
JURUSAN MATEMATIKA
FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM
UNIVERSITAS BRAWIJAYA
MALANG

2008


UNIVERSITAS BRAWIJAYA

LEMBAR PENGESAHAN SKRIPSI

PENGARUH WAKTU PENAMBAHAN JOB BARU PADA PENJADWALAN JOB SHOP DINAMIS MENGGUNAKAN ALGORITMA GENETIK

Oleh:
HASAN BASRI
0310960037-96

Setelah dipertahankan di depan Majelis Penguji
Pada tanggal 9 Juni 2008
dan dinyatakan memenuhi syarat untuk memperoleh
gelar Sarjana dalam bidang Ilmu Komputer

Pembimbing I	Pembimbing II
Drs. Achmad Ridok, M.Kom	Bayu Rahayudi, ST., MT
NIP. 132 090 392	NIP. 132 318 424

Mengetahui,
Ketua Jurusan Matematika
Fakultas MIPA Universitas
Brawijaya

Dr. Agus Suryanto, M.Sc
NIP. 132 126 049

LEMBAR PERNYATAAN

Saya yang bertanda tangan di bawah ini :

Nama **:** Hasan Basri
NIM **:** 0310960037-96
Jurusan **:** Matematika
Penulis skripsi berjudul **:**Pengaruh Waktu Penambahan Job Baru Pada Penjadwalan Job Shop Dinamis Menggunakan Algoritma Genetik.

Dengan ini menyatakan bahwa :

1. **Isi dari skripsi yang saya buat adalah benar-benar karya sendiri dan tidak menjiplak karya orang lain, selain nama-nama yang termaktub di isi dan tertulis di daftar pustaka dalam skripsi ini.**
2. **Apabila dikemudian hari ternyata skripsi yang saya tulis terbukti hasil jiplakan, maka saya akan bersedia menanggung segala resiko yang akan saya terima.**

Demikian pernyataan ini dibuat dengan segala kesadaran.

Malang, 9 Juni 2008
Yang menyatakan,

(Hasan Basri)
NIM. 0310960037

PENGARUH WAKTU PENAMBAHAN *JOB* BARU PADA PENJADWALAN *JOB SHOP* DINAMIS MENGGUNAKAN ALGORITMA GENETIK

ABSTRAK

Penjadwalan *job shop* dinamis terjadi, jika saat proses produksi sedang berlangsung, ada penambahan *job* baru yang menunggu untuk segera diselesaikan. Terhadap permasalahan tersebut Fang mengemukakan dua alternatif penyelesaian, yang pertama dengan mengabaikan jadwal lama dan membuat jadwal yang baru dan yang kedua dengan mempertahankan jadwal lama dan menjadwalkan kembali operasi-operasi yang belum diselesaikan. Pada penelitian ini kedua alternatif penyelesaian tersebut, diimplementasikan dengan menggunakan algoritma genetik. Fokus penelitian ini adalah untuk mengetahui pengaruh waktu penambahan *job* baru terhadap *makespan* yang dihasilkan oleh kedua pendekatan yang dikemukakan oleh Fang.

Uji coba program penjadwalan *job shop* dinamis dilakukan dengan beberapa kombinasi jumlah total *job*, jumlah mesin, jumlah *job* pertama dan waktu penambahan *job* baru. Dari hasil uji coba, diketahui bahwa pendekatan pertama memiliki nilai *mekespan* yang relatif tetap seiring dengan bertambahnya waktu penambahan *job* baru. Hal ini terjadi karena yang menjadi pokok permasalahan pada pendekatan pertama adalah job baru, sehingga jadwal awal tak berpengaruh. Sedangkan pendekatan kedua memiliki nilai *makespan* yang relatif semakin besar dengan bertambahnya waktu penambahan *job* baru. Hal ini terjadi karena dengan bertambahnya waktu penambahan *job* baru, berarti semakin banyak operasi pada jadwal awal yang selesai dikerjakan, dengan kata lain waktu menganggur mesin semakin besar yang berakibat pada meningkatnya *makespan*.

The Effect Of New *Job* Time Increasing Toward Dynamic *Job Shop* Scheduling Using Genetic Algorithm

Abstrack

Dynamic *job shop* scheduling occurred when the producing proses is on, there is new *job* addition waits for to be worked out. Due to their problem, Fang came out for two alternative solution, the first is by ignoring the old schedule and replace it with a new schedule. The second solution is by leaving the old schedule and reschedule the undone operations. In this research, two of those solution are implement by using genetic algorithm. This research is focus on purpose to know the effect of new *job* time increasing toward *makespan*, obtained from those Fang's two approaching solution.

Testing process of the dynamic *job shop* scheduling program , uses some combinations of the total amount of *job*, number of machine, number of first *job* and time addition of new *job*. The testing result that the first approachment come out with *makespan* value in sequence of relative values by the increasing of new *job* times. This happens because the first approachment is focus on problem of new *job*, so that the earlier schedule hasn't a real big impact while, the second approachment has *makespan* value in a relatively increasing value by time new *job* time increase. This happens because of by the time increase, means that the more number of operation done on the earlier schedule, in the other words the idle time of machine is growing bigger that influence the increasing of *makespan*.

KATA PENGANTAR

Puji syukur penulis panjatkan kehadirat Allah SWT, karena atas segala rahmat dan limpahan hidayahnya, Skripsi yang berjudul "*Pengaruh Waktu Penambahan Job Baru Pada Penjadwalan Job Shop Dinamis Menggunakan Algoritma Genetik*" ini dapat diselesaikan. Skripsi ini disusun dan diajukan sebagai syarat untuk memperoleh gelar sarjana pada program studi Ilmu Komputer, Jurusan Matematika, Fakultas MIPA, Universitas Brawijaya.

Dalam penyelesaian skripsi ini, penulis telah mendapat begitu banyak bantuan dari banyak pihak. Atas bantuan yang telah diberikan, penulis ingin menyampaikan penghargaan dan ucapan terima kasih kepada:

1. Drs. Ahmad Ridok, M.Kom selaku pembimbing pertama. Terima kasih atas semua saran, bantuan, kritikan, waktu, dorongan semangat dan bimbingannya.
2. Bayu Rahayudi, ST.,MT selaku pembimbing kedua. Terima kasih atas kesediaannya membimbing dalam penulisan skripsi ini sehingga mudah dan enak dibaca.
3. Wayan F. Mahmudy, S.Si., MT selaku Ketua Program Studi Ilmu Komputer Universitas Brawijaya Malang.
4. Dian Eka R. S.Si.,M.Kom selaku penasehat akademik yang banyak memberikan masukan demi kelancaran perkuliahan.
5. Bapak, ibu dan saudaraku yang senantiasa berdoa dan memberi dukungan dan semangat.
6. Semua teman-teman Ilmu Komputer angkatan 2003 dan 2004 terima kasih atas dukungan dan doanya.
7. Semua pihak lain yang telah membantu hingga terselesaikannya penyusunan penulisan tugas akhir ini.

Semoga penulisan skripsi ini bermanfaat bagi pembaca sekalian. Penulis menyadari bahwa skripsi ini masih jauh dari kesempurnaan sehingga penulis mengharapkan masukan dan saran yang membangun dari pembaca.

Malang, 7 Mei 2008

Penulis

DAFTAR ISI

BAB I
PENDAHULUAN

1.1 Latar Belakang

Penjadwalan produksi merupakan komponen yang sangat penting dalam sebuah proses produksi, karena dengan penjadwalan produksi yang optimal diharapkan proses produksi dapat berjalan dengan lebih efisien. Untuk memaksimalkan penggunaan keseluruhan mesin, perlu dilakukan penjadwalan kerja yang tepat bagi setiap mesin. Dimana obyektif dari permasalahan penjadwalan tersebut adalah untuk mendapatkan sebuah jadwal kerja mesin yang paling minimal dari sisi waktu produksi (Noversada, 2006).

Penjadwalan *Job shop* adalah menjadwalkan proses produksi dari masing-masing **n** *job* yang mempunyai urutan proses produksi dan melalui **m** mesin yang berbeda, dengan tujuan untuk menentukan urutan dari *job* yang ada agar dapat memberikan hasil yang optimal.

Terdapat dua macam penjadwalan *job shop* berdasarkan waktu kedatangannya yaitu *job shop* statis dan *job shop* dinamis. *Job shop* statis terjadi apabila saat proses produksi sedang berlangsung, tidak terjadi penambahan *job* baru. Sedangkan *job shop* dinamis yaitu ketika proses produksi sedang berlangsung sesuai dengan jadwal yang telah disusun berdasarkan *job* yang ada kemudian terjadi penambahan sejumlah *job* baru yang menunggu untuk dikerjakan.

Ada dua pendekatan penjadwalan ulang guna menangani permasalahan *job shop* dinamis yang dikemukakan oleh Fang (1994) yaitu: pertama dengan menjadwalkan kembali dari awal, dan kedua meneruskan beberapa operasi yang telah dikerjakan pada jadwal lama dan melakukan penjadwalan ulang terhadap sisa operasi yang belum dikerjakan. Pendekatan pertama dilakukan dengan membuang jadwal lama dan membuat jadwal baru dari awal dengan data yang baru. Pendekatan kedua tetap mempertahankan sebagian jadwal yang telah disusun sebelumnya dan menyusun jadwal baru dari sisa *job-job* yang belum selesai dikerjakan.

Metode-metode dalam penjadwalan ada bermacam-macam. Metode paling optimal adalah metode matematis yang menguji semua kemungkinan solusi dan mengambil solusi yang paling optimal. Namun karena metode ini memerlukan waktu yang lama dan sulit dilakukan sehingga menghabiskan banyak waktu dan biaya, maka muncul yang disebut metode *heuristik*. Algoritma genetik

sebagai salah satu metode *heuristik* dipilih dengan pertimbangan bahwa sejak pertama kali dirintis oleh Jhon Hollan pada 1960-an, algoritma genetik telah dipelajari, diteliti dan diaplikasikan secara luas pada berbagai bidang ilmu. Algoritma genetik banyak digunakan pada masalah praktis yang berfokus pada pencarian parameter-parameter optimal (optimasi). Keunggulan algoritma genetik sangatlah jelas terlihat dari kemudahan implementasi dan kemampuannya menentukan solusi yang bagus (bisa diterima) secara tepat untuk masalah-masalah berdimensi tinggi (Suyanto 2005).

Pendekatan penjadwalan ulang dengan mempertahankan sebagian jadwal yang dikemukakan oleh Fang ini, sebelumnya pernah diimplementasikan oleh Christian Bierwirth dan Dirk C. Mattfeld (1999) dimana disimpulkan bahwa pendekatan dengan mempertahankan sebagian jadwal tersebut bisa digunakan sebagai alternatif penyelesaian untuk permasalahan *jobshop* dinamis. Kemudian jurnal yang dibuat oleh Christian Bierwirth dan Dirk C. Mattfeld tersebut ditulis ulang oleh Nico Saputro, Yento (2003). Pada skripsi ini bukan hanya pendekatan dengan mempertahankan sebagian jadwal saja yang digunakan tetapi kedua pendekatan yang dikemukakan oleh Fang diimplementasikan sekaligus, kedua pendekatan itu yang pertama adalah menjadwalkan ulang dengan mengabaikan jadwal lama (jadwal awal sebelum ada penambahan *job*) kemudian menjadwalkan kembali semua *job* dari awal, yang kedua adalah menjadwalkan ulang dengan mempertahankan sebagian jadwal yang ada. Pada skripsi ini dilakukan penelitian pengaruh waktu penambahan *job* baru terhadap *makespan* (waktu yang dibutuhkan untuk menyelesaikan semua operasi) yang dihasilkan oleh masing-masing pendekatan. Hal ini dilakukan karena dalam jurnal yang disusun oleh Fang belum memberikan keterangan mengenai hal tersebut. Untuk melakukan penelitian pengaruh waktu penambahan *job* baru terhadap *makespan* yang dihasilkan digunakan algoritma genetik.

1.2 Rumusan Masalah

Rumusan masalah dalam tugas akhir ini adalah:

1. Bagaimana memodelkan dan mengimplementasikan algoritma genetik untuk pendekatan pertama yaitu dengan mengabaikan jadwal awal.

2. Bagaimana memodelkan dan mengimplementasikan algoritma genetik untuk pendekatan kedua yaitu dengan hanya menjadwal ulang terhadap operasi yang belum dikerjakan.
3. Bagaimana pengaruh waktu penambahan *job* baru terhadap *makespan* yang dihasilkan oleh kedua pendekatan.

1.3 Tujuan

Tujuan yang ingin dicapai dari pembuatan skripsi ini adalah mengimpelementasikan model algoritma genetik yang dibuat dan mengetahui apa pengaruh waktu penambahan *job* baru, terhadap *makespan* yang dihasilkan oleh kedua pendekatan yang dikemukakan oleh Fang.

1.4 Batasan Masalah

Batasan masalah dalam penulisan tugas akhir ini adalah:

1. Pada waktu yang sama tiap mesin hanya boleh memproses satu operasi.
2. Tiap mesin hanya dapat mengerjakan satu jenis pekerjaan saja.
3. Pada setiap *job* waktu pengerjaan operasi dan mesin-mesin yang dibutuhkan telah diketahui .
4. Waktu perpindahan antar mesin diabaikan.
5. Pengerjaan tiap operasi tidak dapat disela oleh operasi yang lain (*non preemtif*).
6. Waktu kedatangan *job* sebanyak dua kali, satu saat produksi belum berlangsung (belum ada jadwal produksi) sedangkan yang satu lagi ketika produksi sedang berlangsung.
7. Data yang digunakan adalah data acak (data *random*).

1.5 Manfaat

Manfaat yang diperoleh dari penulisan tugas akhir ini adalah, memberikan gambaran mengenai pengaruh waktu penambahan *job* baru terhadap *makespan* yang diperoleh, dari implementasi kedua pendekatan yang dikemukakan oleh Fang dengan menggunakan algoritma genetik, berdasarkan spesifikasi perangkat keras dan perangkat lunak yang digunakan.

1.6 Metodologi Pemecahan Masalah

Untuk mencapai tujuan yang dirumuskan sebelumnya, maka metodologi yang digunakan dalam penulisan tugas akhir ini adalah:

1. Studi Literatur
   Mempelajari teori yang berhubungan dengan algoritma genetik dan *job shop* dari berbagai sumber.
2. Pendefinisian dan Analisis Masalah.
   Mendefinisikan dan menganalisis masalah untuk mencari solusi yang tepat.
3. Perancangan dan Implementasi Sistem
   Membuat perancangan perangkat lunak dan mengimplementasikan hasilnya untuk membuat perangkat lunak tersebut.
4. Uji Coba dan Analisis Hasil Implementasi
   Menguji coba perangkat lunak tersebut dan menganalisa hasil dari implementasi tersebut apakah sudah sesuai dengan tujuan yang dirumuskan sebelumnya, untuk kemudian dievalusi dan disempurnakan.

1.7 Sistematika Penulisan

Sistematika penulisan tugas akhir ini dibagi menjadi lima bab dengan masing-masing bab diuraikan sebagai berikut:

1. BAB I PENDAHULUAN
   Berisi latar belakang masalah, perumusan masalah, batasan masalah, tujuan penulisan, manfaat penulisan, metodologi pemecahan masalah, dan sistematika penulisan.
2. BAB II TINJAUAN PUSTAKA
   Bab ini membahas mengenai teori-teori penunjang yang berkenaan dengan konsep dasar *job shop*, dan dasar-dasar algoritma genetik yang akan digunakan untuk menyelesaikan permasalahan *job shop*.
3. BAB III METODE DAN PERANCANGAN
   Bab ini membahas mengenai penggunaan teori-teori dalam perancangan perangkat lunak ini. Serta membahas bagaimana perangkat lunak dan penelitian ini direncanakan, dan diimplementasikan.

4. BAB IV HASIL DAN PEMBAHASAN
   Pada bab ini akan dilakukan implementasi sistem, serta analisa hasil uji coba pada kedua pendekatan yang diungkapkan oleh Fang.
5. BAB V PENUTUP
   Bab ini berisi kesimpulan dari hasil uji coba dan saran untuk pengembangan penelitian lebih lanjut.

BAB II
TINJAUAN PUSTAKA

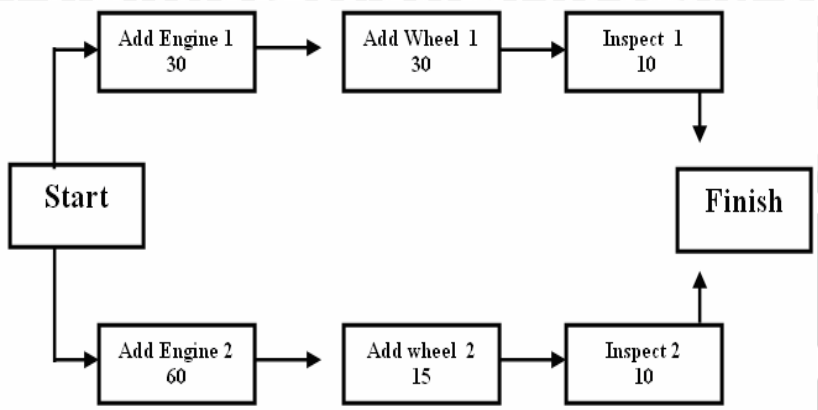
2.1 *Job shop*

2.1.1 Penjadwalan Produksi

Penjadwalan didefinisikan sebagai pengalokasian sumber-sumber daya selama suatu rentang waktu untuk melakukan sekumpulan tugas. Tujuan penjadwalan adalah mengoptimalisasi penggunaan sumber daya sehingga tujuan produksi dapat tercapai (Dian, 2005). Sumber lain menyebutkan bahwa tujuan utama dari proses penjadwalan adalah menentukan waktu suatu operasi mulai dikerjakan. Hal ini dilakukan pada seluruh operasi sampai seluruh operasi sudah dijadwalkan dan memenuhi setiap batasan masalah yang dirumuskan. (Gamma, 2006).

Masalah penjadwalan memiliki berbagai variasi *constraint* yang harus dipenuhi dalam pencarian solusi. Berbagai algoritma penjadwalan sudah dikemukakan yang ditujukan untuk menyelesaikan masalah penjadwalan tertentu. Algoritma penjadwalan yang paling sederhana merupakan algoritma dasar yang menentukan kapan suatu operasi dikerjakan tanpa memperhatikan *resource* yang diperlukan untuk mengerjakan operasi tersebut Sebagai contoh, masalah penjadwalan sederhana direpresentasikan dalam *graf* seperti pada Gambar 2.1. Pada gambar tersebut terdapat dua *job* perakitan mobil. Setiap *job* terdiri dari tiga operasi yaitu, *AddEngine*, *AddWheels*, dan *Inspect*. Setiap operasi tersebut memiliki durasi yang berbeda-beda. Durasi setiap operasi merupakan angka yang tercantum pada setiap simpul *graf* yang merepresentasikan operasi tersebut.

Anak panah pada *graf* tersebut melambangkan *precedence* setiap operasi. Operasi yang berada di kiri anak panah harus dikerjakan lebih dahulu dari operasi yang berada di kanan anak panah. Pada Gambar 2.1 terdapat operasi *Start* dan *Finish* yang sebenarnya merupakan operasi semu (*dummy*) yang ditambahkan untuk menyatakan bahwa kedua *job* tersebut berada pada satu masalah penjadwalan. Durasi kedua operasi semu ini sama dengan nol (Gamma, 2006).



Gambar 2.1 Penjadwalan perakitan mobil

Karakteristik mesin secara umum ada dua jenis, pertama satu mesin memiliki kemampuan untuk melakukan beberapa operasi dengan waktu proses tiap operasi yang berbeda-beda. Yang kedua satu mesin hanya bisa melakukan satu operasi saja, pada skripsi ini penulis menggunakan ketentuan bahwa satu mesin hanya dapat melakukan satu operasi saja (Gregorius .2003). Keterangan lebih lengkapnya dapat dilihat pada Tabel 2.1 dan Tabel 2.2 berikut.

Tabel 2.1 Ilustrasi satu mesin dapat mengerjakan beberapa operasi

Mesin	Operasi yang dapat dilakukan
M1	O1, O3, O4
M2	O2, O3, O5, O6
M3	O1, O2

Tabel 2.2 Ilustrasi satu mesin hanya melakukan satu operasi saja

Mesin	Operasi yang dapat dilakukan
M1	O1
M2	O3
M3	O2

Pada Tabel 2.1 dapat dilihat bahwa mesin 1 punya kemampuan melakukan operasi 1, operasi 3 dan operasi 4. Sedangkan pada Tabel 2.2 mesin 1 hanya dapat mengerjakan operasi 1 saja.

2.1.2 Penjadwalan *Job shop*

Penjadwalan *job shop* adalah penjadwalan sekumpulan *job* (J) dan set mesin (M). Dimana setiap *job* tersusun atas serangkaian operasi (O). Setiap operasi harus dikerjakan pada mesin yang spesifik pada rentang waktu yang ditentukan tanpa bisa disela oleh operasi yang lainnya. Tak satupun mesin yang diperkenankan mengerjakan lebih dari satu operasi pada kurun waktu yang sama dan setiap operasi harus telah diselesaikan sebelum mengerjakan operasi berikutnya yang menjadi bagian dari sebuah *job* (Keith Schmidt, 2001). Dalam permasalahan *job shop* terdapat beberapa pekerjaan yang memiliki operasinya sendiri-sendiri yang hanya dapat dijalankan di sebuah mesin pada suatu waktu tertentu. Objektif dari permasalahan *job shop* ini adalah untuk mendapatkan sebuah jadwal operasi pekerjaan yang memakan waktu yang paling singkat yang mungkin dilakukan (Noversada, 2006).

Fungsi objektif permasalahan *job shop* biasa disebut *makespan*, dengan kata lain tujuan penjadwalan *job shop* adalah meminimumkan *makespan*. Tiap operasi memiliki waktu mulai dan waktu selesai, waktu mulai didapatkan dari waktu terakhir operasi sebelumnya selesai dikerjakan. Sedangkan waktu selesai adalah waktu mulai dan lama waktu yang dibutuhkan operasi tersebut selesai dikerjakan. *Makespan* adalah total waktu yang dibutuhkan oleh semua operasi untuk dapat selesai dikerjakan (Megan, 2004). Sumber lain menyebutkan bahwa *makespan* adalah waktu penyelesaian produk (Hetti.M.N dkk, 2006).

Berdasarkan waktu kedatangannya penjadwalan *job shop* dibagi menjadi dua jenis yaitu *job shop* statis dan *job shop* dinamis. *Job shop* statis adalah *job shop* dimana semua *job* datang secara bersamaan dan tak terjadi penambahan ketika proses produksi sedang berlangsung sehingga jadwal produksi yang disusun berdasarkan data awal kedatangan *job shop*. Sedangkan *job shop* dinamis adalah *job shop* dimana ketika proses produksi sedang berlangsung sesuai dengan jadwal yang telah ditentukan sebelumnya kemudian terjadi penambahan sejumlah *job* baru yang harus dikerjakan dimana kedatangan *job* tersebut seringkali tidak dapat diramalkan (*non*

deterministik) (Nico, 2003). Ada dua pendekatan penjadwalan ulang guna menangani permasalahan tersebut yaitu: dengan menjadwalkan kembali dari awal, atau meneruskan beberapa operasi yang telah dikerjakan pada jadwal lama dan melakukan penjadwalan ulang terhadap sisa operasi yang belum dikerjakan.

Pendekatan pertama dilakukan dengan cara membuang jadwal lama dan membuat jadwal baru dari awal dengan data yang baru. Pendekatan pertama dilakukan dengan tujuan agar proses produksi benar-benar berjalan dengan maksimal, hanya saja pendekatan pertama ini mengabaikan atau mengorbankan operasi-operasi yang sudah dan sedang dikerjakan. Berbeda dengan pendekatan pertama pendekatan kedua dilakukan dengan tetap mempertahankan sebagian jadwal yang telah disusun sebelumnya dan menyusun jadwal baru dari sisa *job-job* yang belum selesai dikerjakan dan *job-job* yang baru datang. Alternatif kedua memiliki kelemahan yaitu memiliki alur penyelesaian yang relatif rumit karena harus mendata ulang operasi mana yang telah dan sedang dikerjakan dan operasi mana yang belum dikerjakan untuk kemudian dilakukan penjadwalan ulang, selain itu besar kemungkinan hasil penjadwalan ulang tak seoptimal jadwal yang dihasilkan oleh cara yang pertama. Tetapi alternatif langkah kedua ini memiliki kelebihan yaitu tidak mengorbankan operasi-operasi yang sedang dan telah berlangsung, (Fang, 1994).

Job shop dinamis menuntut untuk segera menangani *job* yang baru datang dengan memasukkannya ke dalam jadwal produksi. Pemodelan *job shop* dinamis ini bisa dipandang seperti penjadwalan *job shop* statis dengan penambahan sisa *job* yang belum selesai diproses dengan *job* yang baru datang. Karena operasi-operasi tak dapat disela maka tiap operasi harus diselesaikan terlebih dahulu sebelum menjalankan operasi yang selanjutnya baik dari *job* yang lama maupun *job* yang baru datang yang telah dijadwalkan. Karena pada penjadwalan *job shop* dinamis waktu kedatangan *job* baru yang tak sama berakibat pada waktu tunggu mesin atau waktu menganggur mesin (*idle time*) berpotensi meningkat (Marko Snoke, 1999).

2.2 Algoritma Genetik

Heuristik adalah sebuah teknik yang meningkatkan efisiensi dari sebuah proses pencarian dengan mengabaikan klaim terhadap kesempurnaan penyelesaian. Metode ini baik karena menunjuk pada

arah atau hasil yang diinginkan, tetapi di lain sisi juga buruk karena ada kemungkinan dilewatinya suatu solusi yang lebih baik. Beberapa teknik heuristik dapat melakukan proses pencarian tanpa mengorbankan klaim terhadap kesempurnaan, sementara teknik–teknik lain dalam heuristik dapat mengakibatkan diabaikannya jalur-jalur terbaik yang mungkin ada. Tetapi rata–rata heuristik dapat meningkatkan kualitas dari rute yang dieksplorasi. Dengan menggunakan heuristik yang baik, dapat diharapkan untuk mendapatkan solusi yang baik (walaupun belum tentu terbaik) pada problem–problem berat dengan waktu yang dibutuhkan kurang dari waktu eksponensial (Gregorius, 2003).

Algoritma genetik merupakan salah satu contoh dari algoritma *heuristik*. Algoritma Genetik merupakan algoritma pencarian yang bekerja berdasarkan mekanisme seleksi alam dan genetika. Pada genetika, kromosom tersusun dari gen-gen. Tiap gen mempunyai sifat tertentu (*allele*), dan posisi tertentu (*locus*). Satu atau lebih kromosom bergabung membentuk paket genetik yang disebut *genotif* . Interaksi *genotif* dengan lingkungannya disebut *fenotif* . Pada algoritma genetik, kromosom berpadanan dengan string dan gen dengan karakter. Setiap karakter mempunyai posisi (*locus*) dan arti tertentu (*allele*). Satu atau lebih string bergabung membentuk struktur (*genotif*), dan bila struktur tersebut di-*decode*-kan akan diperoleh salah satu alternatif solusi (*fenotif*).

Kemunculan algoritma genetik diinspirasikan dari teori-teori dalam ilmu biologi, sehingga banyak istilah dan konsep biologi yang digunakan dalam algoritma genetik. Sesuai dengan namanya, proses-proses yang terjadi dalam algoritma genetik sama dengan apa yang terjadi pada evolusi biologi. Dalam ilmu biologi individu disebut spesies yang hidup, bereproduksi dan akhirnya mati pada suatu tempat yang disebut populasi. Jika anggota populasi terpisah entah karena banjir, gempa dan sebagainya maka mereka akan membentuk populasi baru. Dalam waktu yang cukup lama mungkin saja akan terjadi proses pembentukan spesies baru yang dikenal dengan nama *speciation*. Dalam hal ini terjadi perubahan hereditas (*heredity*) secara bertahap yang membentuk ciri-ciri baru pada spesies tersebut. Sebagai contoh, spesies pemangsa mengalami perubahan bertahap sehingga memiliki gigi taring yang lebih panjang dan tajam. Hal ini terjadi akibat evolusi yang terjadi pada mangsa yang memiliki kulit

yang semakin tebal dan keras. Perubahan bertahap secara bersamaan pada kedua spesies disebut sebagai *co-evolution.*

Konsep yang penting disini adalah hereditas, yaitu sebuah ide yang menyatakan bahwa sifat-sifat individu dapat dikodekan dengan cara tertentu sehingga sifat tersebut dapat diturunkan kepada generasi berikutnya. Bagaimana informasi ini dapat diturunkan dan disimpan dalam individu ? Jawabannya adalah bahwa setiap individu dalam satu spesies membawa sebuah *genome* yang berisi beberapa kromosom dalam bentuk molekul-molekul DNA. Setiap kromosom berisi sejumlah *gen*, dimana unit-unit hereditas dan pengkodean informasi diperlukan untuk membangun dan menjaga suatu individu. Jadi informasi disimpan dalam suatu pola digital, selama perkembangan dan selama kehidupan. DNA dibaca dengan suatu enzim yang disebut *RNA polymerese*.

Konsep penting dari teori evolusi adalah *fitness* dan *selection* untuk proses produksi. Pada proses evolusi di dunia nyata, terdapat dua cara reproduksi yaitu sexsual dan asexsual. Pada reproduksi sexsual , kromosom-kromosom dari dua individu dikombinasikan untuk menyusun individu baru. Artinya kromosom pada individu baru berisi beberapa gen yang diambil dari *parent* pertama dan selebihnya didapatkan dari *parent* kedua. Hal ini disebut sebagai *crossover* (pindah silang). Namun demikian proses pengkopian gen *parent* ini tidak luput dari kesalahan. Kesalahan pengkopian gen ini dikenal dengan istilah mutasi (*mutation*). Sedangkan pada reproduksi aseksual hanya satu individu *parent* yang diperhatikan, sehingga tidak terjadi proses *crossover*, tetapi proses mutasi juga mungkin terjadi.

Sejak pertama kali dirintis oleh Jhon Hollan pada 1960-an, algoritma genetik telah dipelajari, diteliti dan diaplikasikan secara luas pada berbagai bidang ilmu. Algoritma genetik banyak digunakan pada masalah praktis yang berfokus pada pencarian parameter-parameter optimal. Keunggulan algoritma genetik sangatlah jelas terlihat dari kemudahan implementasi dan kemampuannya menentukan solusi yang bagus (bisa diterima) secara tepat untuk masalah-masalah berdimensi tinggi (Suyanto, 2005).

2.2.1 Komponen-Komponen Algoritma Genetik

Pada dasarnya algoritma genetik memiliki lima komponen, tetapi banyak metode yang bervariasi yang diusulkan pada masing-

masing komponen tersebut. Masing-masing metode memiliki kelebihan dan kekurangan. Suatu metode yang bagus untuk menyelesaikan masalah A belum tentu bagus untuk menyelesaikan masalah B, atau bahkan tak dapat digunakan untuk menyelesaikan masalah C. Untuk lebih jelasnya akan diuraikan sebagai berikut:

2.2.1.1 Pengkodean Kromosom

Pada dasarnya pengkodean kromosom disesuaikan dengan masalah yang akan diselesaikan. Syarat yang harus dipenuhi dalam pengkodean kromosom adalah dapat mewakili dari kondisi atau permasalahan yang akan diselesaikan. Secara umum ada yang menggunakan bantuan kode biner yang nantinya harus dikonversi dulu kedalam bentuk angka desimal dan ada pula yang langsung menggunakan angka desimal. Untuk lebih jelasnya dapat dilihat pada Tabel 2.3 dan Tabel 2.4 (Fang, 1994).

Tabel 2.3 Contoh kromosom yang menggunakan angka desimal.

index	***chromosome***
1	3 8 4 8 6 7
2	7 3 7 6 1 3
3	5 3 5 6 5 8
4	7 6 7 2 4 5
5	1 7 4 5 2 3

Tabel 2.4 Contoh kromosom yang menggunakan angka biner

index	***chromosome***
1	0 1 0 0 1 0 0 1 0 1 0
2	0 1 0 0 1 1 0 0 0 0 1
3	0 1 0 0 1 1 0 0 0 1 0
4	0 0 1 0 0 1 1 0 1 1 1
5	0 0 1 0 0 1 1 0 0 0 0

2.2.1.2 Penghitungan Nilai *Fitness*

Suatu individu dievaluasi berdasarkan suatu nilai *fitness* tertentu sebagai ukuran performasinya. Di dalam evolusi alam individu yang bernilai *fitness* tinggi yang akan bertahan hidup.

Sedangkan individu yang bernilai *fitness* rendah akan mati. Pada masalah optimasi, jika solusi yang dicari adalah memaksimumkan sebuah fungsi *h* (dikenal sebagai masalah maksimasi) maka, nilai *fitness* yang dihitung adalah nilai dari fungsi *h* tersebut, yaitu f = *h* (dimana f adalah nilai *fitness*). Tetapi jika masalahnya adalah meminimalkan fungsi *h* (masalah minimasi), maka fungsi *h* tidak bisa digunakan secara langsung. Oleh karena itu nilai *fitness* yang bisa digunakan adalah f = 1/*h*, yang artinya semakin kecil nilai *h* maka semakin besar nilai f nya. Namun akan terjadi masalah jika nilai *h* adalah 0 yang berakibat nilai f tak berhingga. Untuk mengatasinya maka *h* harus ditambah dengan suatu nilai a (suatu nilai yang sangat kecil) sehingga nilai *fitness* menjadi f = 1/(*h*+a). Penentuan nilai *fitness* sangat berpengaruh pada performasi algoritma genetik secara keseluruhan. Pada permasalahan *Jobshop* ini nilai fungsi yang dimaksud adalah *makespan* (waktu yang dibutuhkan untuk menyelesaikan seluruh produk), dan optimasi yang dimaksud adalah minimasi, sehingga semakin besar nilai *makespan*nya berarti semakin kecil nilai *fitness*nya. Pada proses *crossover* individu dengan nilai *fitness* tinggi punya peluang yang lebih besar untuk terpilih sebagai *parent*. Dalam beberapa kasus nilai *fitness* sangat sederhana dan bisa ditemukan dengan mudah, tetapi pada beberapa kasus nilai *fitness* kompleks dan sulit ditemukan (Suyanto, 2005).

2.2.1.3 Seleksi *Parent*

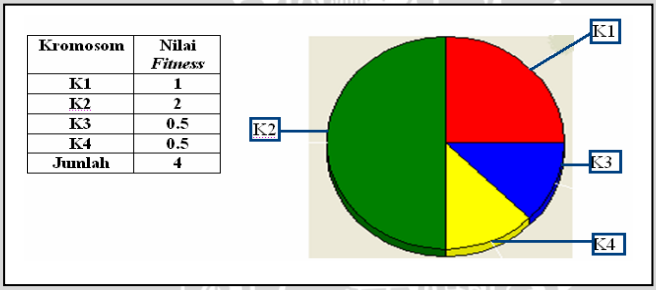
Pemilihan dua buah kromosom sebagai orang tua (*parent*) yang akan dipindah silangkan biasanya dilakukan secara proporsional sesuai dengan nilai *fitness*nya. Suatu metode seleksi yang umum dilakukan adalah *roulette wheel* (roda *roulette*). Sesuai dengan namanya, metode ini menirukan permainan *roulette wheel* dimana masing-masing kromosom menempati potongan lingkaran pada roda *roulette* secara proporsional sesuai dengan nilai *fitness*nya seperti yang ada pada Gambar 2.2. Kromosom yang memiliki nilai *fitness* besar menempati potongan lingkaran yang lebih besar dibandingkan dengan kromosom yang memiliki nilai *fitness* lebih kecil.

Untuk selengkapnya langkah-langkahnya adalah :

1. Hitung nilai *fitness* untuk setiap kromosom
2. Hitung total nilai *fitness* pada populasi

3. Hitung probabilitas seleksi pada setiap kromosom
4. Hitung probabilitas kumulatif untuk setiap kromosom

Turnament Selection adalah alternatif lain untuk pemilihan *parent*. Pada seleksi alam yang terjadi di dunia nyata, beberapa individu (biasanya individu jantan) berkompetisi dalam sebuah kelompok kecil sampai tersisa hanya satu individu pemenang. Individu pemenang inilah yang memungkinkan melakukan perkawinan. Dalam bentuk paling sederhana metode ini mengambil dua kromosom secara acak dan kemudian menyeleksi salah satu yang bernilai *fitness* paling tinggi untuk menjadi *parent* pertama. Cara yang sama dilakukan untuk mendapatkan *parent* kedua (Suyanto, 2005), ilustrasi seleksi berdasarkan *roulette whell* pada Gambar 2.2.



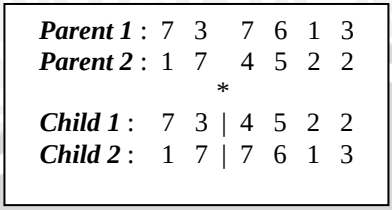
Gambar 2.2 Ilustrasi *roulette wheel*

2.2.1.4 *Crossover* (pindah silang)

Salah satu komponen paling penting dalam algoritma genetik adalah *crossover* (pindah silang). Sebuah kromosom yang mengarah pada solusi yang optimal bisa diperoleh dari proses *crossover* dua buah kromosom. Jenis *crossover* beraneka macam diantaranya:

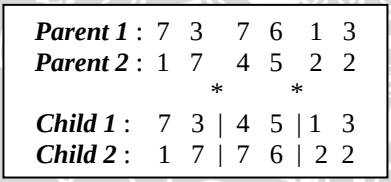
1) *Crossover* dengan satu titik potong

   Yaitu dengan menentukan titik perpotongan tunggal pada kromosom kemudian menentukan bahwa kromosom yang berada di belakang titik perpotongan akan mewarisi sifat dari *parent* pertama sedangkan selebihnya didapat dari *parent* kedua, contoh penggunaannya pada Gambar 2.3.



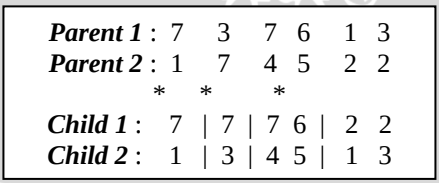
Gambar 2.3 *Crossover* dengan satu titik potong

2) *Crossover* dengan dua titik potong
Seperti pada *crossover* dengan satu titik potong, hanya saja disini terdapat dua perpotongan sehingga anak mewarisi sifat dari kedua orangtua secara bergantian. Awalnya anak mewarisi sifat dari *parent* pertama setelah melewati batas perpotongan pertama anak mewarisi sifat dari *parent* kedua begitu pula bila melewati batas perpotongan kedua anak kembali mewarisi sifat dari *parent* kedua, contoh penggunaannya pada Gambar 2.4.



Gambar 2.4 *Crossover* dengan dua titik potong

3) *Crossover* N titik potong
Ide dasarnya sama dengan *crossover* dengan dua titik potong hanya saja disini terdapat lebih dari dua titik potong sehingga tiap melewati titik potong maka bergantian mewarisi sifat dari kedua *parent*nya, contoh penggunaannya pada Gambar 2.5.



Gambar 2.5 *Crossover* dengan N titik potong

4) *Uniform Crossover*

Intinya dengan membangkitkan bilangan random antar [0 1] (0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0) dimana bila tiap-tiap gen mendapatkan nilai dibawah 0.5 maka anak mewarisi sifat dari *parent* pertama sedangkan jika mendapatkan nilai diatas 0.5 maka anak mewarisi sifat dari *parent* kedua, contoh penggunaannya pada Gambar 2.6.

Parent 1 :	7	*3	*7	6	*1	3
Parent 2 :	1	*7	*4	5	*2	2
Child 1 :	7	7*	4*	6	2*	3
Child 2 :	1	3*	7*	5	1*	2

Gambar 2.6 Uniform *Crossover*

5) *Precedence Preservative Crossover* (PPX)

Precedence Preservative Crossover (PPX) idenya mirip uniform *crossover* hanya saja tidak menggunakan bilangan dengan rentang antara [0 1] tapi langsung menentukan gen mana yang akan mewarisi sifat dari induk pertama dan mana yang akan mewarisi sifat dari induk kedua (Bierwirth and Fang, 1994), contoh penggunaannya pada Gambar 2.7.

Parent permutation 1	A	B	C	D	E	F
Parent permutation 2	C	A	B	F	D	E
Select parent no.(1/2)	1	2	1	1	2	2
Offspring permutation	A	C	B	B	F	E

Gambar 2.7 Precedence Preservative *Crossover* (PPX)

6) *Order Crossover* (OX)

Order Crossover dilakukan dengan melakukan pemilihan sekumpulan gen pada kromosom pertama secara random untuk diturunkan pada anak (*offspring*). Sisa dari gen anak didapat dari gen *parent* kedua. Pada *crossover* ini gen yang diambil dari gen *parent* kedua dengan memperhatikan gen yang diturunkan oleh gen pertama seperti yang dapat dilihat pada Gambar 2.8. Gen

tersebut bilamana sudah diturunkan oleh *parent* pertama tak boleh lagi muncul sehingga digantikan oleh gen dengan urutan selanjutnya. Pada masalah *job shop* yang diperhatikan adalah jumlah operasi bilamana jumlah operasinya sudah melebihi, maka digantikan dengan operasi selanjutnya.

Parent 1	1	2	**3**	**4**	**5**	**6**	7	8
Offspring	7	1	**3**	**4**	**5**	**6**	2	8
Parent 2	5	7	4	1	3	6	2	8

Gambar 2.8 Ilustrasi order *crossover*

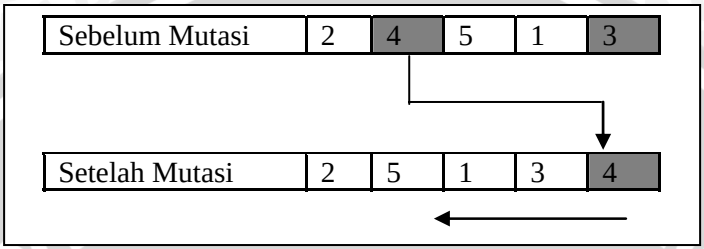
2.2.1.5 Mutasi

Beberapa prosedur mutasi yang sederhana diantaranya dengan memilih dua gen secara acak kemudian saling mempertukarkannya (*Reciprocal Exchange Mutasi*). Hal ini akan mengubah susunan gen dengan sendirinya. Tujuan sebenarnya dari mutasi adalah sedapat mungkin keseluruhan dari kemungkinan solusi dapat muncul karena dengan *crossover* bisa jadi fungsi dengan kemunculan sifat tertentu dapat tertutupi dan terlewatkan. Sehingga solusi yang didapat hanya berdasarkan besar peluang yang telah ditetapkan pada metode *roulette wheel* saja. Alternatif mutasi kedua adalah dengan menentukan gen secara acak kemudian disisipkan pada gen yang lain sedangkan gen sisa digeser letaknya. Alternatif yang lain yaitu memilih secara acak gen hasil *crossover* kemudian membangkitkan angka random antara [0 1] bila angka yang muncul dibawah *mutation rate* (misal ditentukan sebesar 0.001) maka gen tersebut akan digantikan dengan angka random yang dibangkitkan (misalkan random antara [1 5]) seperti pada Gambar 2.9 (Fang, 1994).

Sebelum Mutasi	2	4	5	1	3
Setelah Mutasi	3	4	5	1	2

Gambar 2.9 Contoh mutasi dengan mempertukarkan gen terpilih

Pada Gambar 2.9 terlihat kromosom pertama gen yang terpilih adalah 2 karena saat dibangkitkan angka random [0 1] melebihi mutation rate maka dilakukan perubahan. Sedangkan gen kedua adalah 3 maka dilakukan penukaran diantara keduanya.



Gambar 2.10 Contoh mutasi dengan menggeser gen terpilih

Pada Gambar 2.10 terdapat urutan gen pada sebuah kromosom yang akan dilakukan operasi mutasi, dengan membangkitkan angka random didapatkan dua tempat yaitu gen kedua (4) dan gen kelima (3). Maka proses mutasi dilakukan dengan menempatkan gen kedua tersebut pada gen kelima sedangkan urutan gen yang lain menempati gen di depannya. Dengan kata lain gen yang semula urutan ke-3 menjadi urutan ke-2, gen urutan ke-4 menjadi urutan ke-3 dan seterusnya.

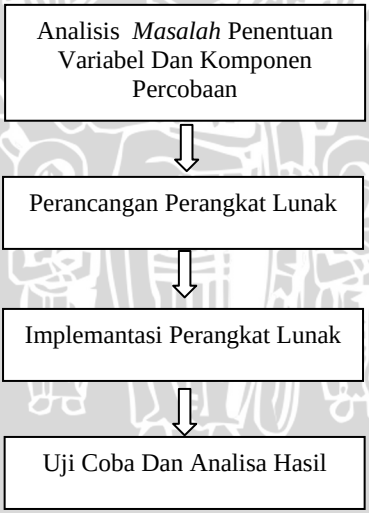
BAB III
METODE DAN PERANCANGAN

Pada bab metode dan perancangan ini akan dibahas langkah-langkah yang digunakan dalam pembuatan perangkat lunak dan metode percobaan.

Tahapan pembuatannya adalah sebagai berikut:

1. Mempelajari metode yang digunakan dari jurnal yang pernah ada, yang telah disinggung pada BAB I dan menganalisa masalah yang dihadapi.
2. Merancang perangkat lunak.
3. Membuat perangkat lunak berdasarkan analisis dan perancangan yang dilakukan.
4. Uji coba perangkat lunak dengan menggunakan data *random*.

Langkah-langkah pembuatan perangkat lunak digambarkan pada Gambar 3.1.



Gambar 3.1 Diagram Alir Pembuatan Perangkat Lunak

3.1 Analisis Sistem

3.1.1 Deskripsi Sistem

Perangkat lunak yang akan dibangun ini merupakan perangkat lunak yang dibuat sebagai implementasi teori yang bertujuan untuk mengetahui bagaimana algoritma genetik menyelesaikan masalah *job shop* dinamis. Perangkat lunak yang dibangun digunakan sebagai alat penguji seberapa besar tingkat kompleksitas *job shop* dan waktu penambahan *job* baru mempengaruhi *makespan* jadwal yang dihasilkan.

3.1.2 Batasan Sistem

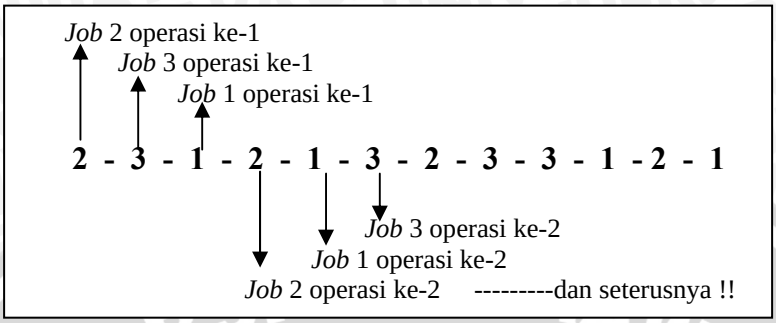
Perangkat lunak ini dibangun untuk menyelesaikan permasalahan *job shop* dinamis yang telah disebutkan di latar belakang. Metode yang digunakan adalah algoritma genetik. Data percobaan yang digunakan adalah data yang dibangkitkan secara *random* dengan tetap memperhatikan syarat dan ketentuan yang berlaku pada penjadwalan *job shop*.

3.2 Perancangan Sistem

Sistem yang akan dibangun ini menggunakan algoritma genetik. Dalam algoritma genetik terdapat beberapa komponen penyusun yang utama yang digunakan untuk memodelkan permasalahan sekaligus digunakan sebagai alat penyelesaian masalah. Pada subbab berikut akan di bahas masing-masing komponen algoritma genetik, diagram alir dan rancangan antarmuka sistem.

3.2.1 Representasi Kromosom

Pada penelitian ini kromosom disusun berdasarkan operasi-operasi yang menyusun masing-masing *job*. Adapun bentuk penulisan kromosom adalah menggunakan nama *job* yang diulang sebanyak jumlah operasi yang dimiliki. Ketentuannya adalah Gen atau nama *job* yang muncul pertama kali mewakili operasi ke-1 pada *job* tersebut begitu pula pada nama *job* yang lain, sehingga saat nama *job* muncul yang kedua kali maka ia mewakili operasi ke-2 pada *job* tersebut begitu seterusnya, lebih jelasnya dapat dilihat pada Gambar 3.2.



Gambar 3.2 Ilustrasi representasi kromosom

Pembacaan arti dari representasi kromosom seperti pada Gambar 3.2, dilakukan ketika dilakukan penghitungan nilai *makespan*. Dengan kata lain ketika proses *crossover* dan mutasi kromosom belum memiliki keterangan mengenai posisi tiap-tiap operasi.

3.2.2 Prosedur Seleksi *Parent*

Terdapat beberapa cara pemilihan *parent* yang digunakan dalam algoritma genetik. Dalam penelitian ini digunakan metode *roulette wheel*, karena metode ini memberikan kesempatan yang besar pada kromosom yang kuat (memiliki nilai *fitness* optimum) untuk dapat disilangkan, dan dari beberapa penelitian sebelumnya metode ini dipandang cukup layak menghasilkan kromosom yang baik (berpeluang menjadi solusi optimal).

3.2.3 Pembentukan Populasi

Pada penelitian ini populasi dibentuk dari gabungan antara *crossover* dengan mutasi. Diharapkan dengan pemilihan operator *crossover* dan mutasi yang baik akan didapatkan individu-individu yang memiliki nilai *fitness* yang tinggi..

3.2.4 Operasi *Crossover*

Untuk optimasi *job shop* ini operator *crossover yang* dipilih yaitu *cut point* (2 titik potong) yaitu dengan memilih secara acak dua titik potong. Gen pada *parent* ke-1 yang berada dibawah titik potong pertama langsung diturunkan pada anak, secara bersamaan gen yang identik dengan anak dihapus dari *parent* ke-2. Selanjutnya

gen pada *parent* ke-2 yang berada antar titik potong ke-1 dan ke2 diturunkkan pada anak, dan yang terakhir gen pada *parent* pertama yang berada setelah titik potong ke-2 diturunkan pada anak, algoritmanya ada pada Gambar 3.3.

```
Bangkitkan bilangan random
If bilangan random> peluang crossover  then do
Begin
 Pilih dua parent
 cari dua titik potong
 for i=1 to panjang kromosom do
   begin
    if i< titik potong ke 1
       child[i]=parent1[i]
    else if i diantara titik potong ke-1  dan
                       titik potong ke-2
       child[i]=parent2[i]
    else
       child[i]=parent1[i]
  end
```

Gambar 3.3 algoritma *crossover*

3.2.5 Operasi Mutasi

Operator mutasi yang dipilih adalah pindah sisip dilakukan dengan cara menyeleksi dua posisi secara random, kemudian gen posisi pertama menempati posisi gen kedua sedangkan posisi gen diantar dua gen terpilih menempati tempat yang ditinggalkan, algoritmanya ada pada Gambar 3.4.

```
Pilih 2 titik potong
b <- kromosom[titik potong ke 1]
for a = titik potong ke 1 to (titik potong ke 2)-1 do
  begin
     kromosom[a] := kromosom[a+1]
  end
kromosom[titik potong ke 2]<- b
```

Gambar 3.4 algoritma mutasi

3.2.6 Penghitungan Nilai *Fitness*

Pada penelitian ini terdapat dua pendekatan yang memiliki cara penghitungan nilai *fitness* yang berbeda. Fungsi *fitness* yang dimaksudkan disini adalah minimasi *makespan* (waktu semua *job* selesai dikerjakan).

a. Penjadwalan dengan mengabaikan operasi yang telah selesai dikerjakan penghitungan nilai *fitness*nya menggunakan langkah-langkah sebagai berikut :
    1. Hitung *makespan* jadwal awal sesuai langkah berikut:
        a) Bentuk suatu *array* berdasarkan kromosom dimana kromosom merupakan nama *job* yang ditulis sebanyak operasi yang dimiliki.
        b) Bentuk *array* baru dengan ketentuan urutan *array* merupakan keterangan operasi ke berapa yang sedang dijadwalkan sesuai masing-masing *job*.
        c) Bentuk lagi *array* yang berisikan keterangan di mesin ke berapa tiap-tiap operasi dikerjakan.
        d) Buat *array* sebanyak jumlah mesin yang berisikan waktu mulai dan waktu selesai mengerjakan operasi sesuai jadwal.
        e) Buat *array* sebanyak jumlah job yang berisikan waktu mulai dan selesai tiap operasi dikerjakan oleh mesin.
        f) Waktu mulai dan waktu selesai mesin pada beberapa kondisi bergantung pada waktu mulai dan selesai *job* yang diproses.
        g) *Makespan* adalah nilai maximum (waktu *finish* tiap mesin) ditambah waktu *Start.*
        h) Nilai *fitness* adalah 1/ *makespan*.
    2. Nilai *fitness* penjadwalan didapatkan dengan menghitung *makespan* operasi-operasi dari *job* yang belum terjadwalkan. Dengan ketentuan waktu *start* tiap mesin adalah waktu *finish* dari tiap operasi pada jadwal awal .

b. Penjadwalan dengan meneruskan operasi yang telah selesai dikerjakan dan menjadwalkan sisa operasi beserta operasi dari *job* baru penghitungan nilai *fitness*nya menggunakan langkah-langkah sebagai berikut :
    1. Menghitung *makespan* jadwal awal sesuai dengan langkah-langkah pada point a .
    2. Melakukan cek pada tiap operasi , operasi mana saja yang dianggap sudah selesai dikerjakan dengan ketentuan memiliki waktu mulai dibawah waktu penambahan *job* baru.
    3. Melakukan penjadwalan terhadap operasi yang belum dikerjakan dan operasi dari *job* baru menggunakan langkah-langkah pada point a.

4. Nilai *fitness* didapatkan dengan menghitung nilai *makespan* jadwal baru, ketentuannya adalah waktu *start* tiap mesin adalah waktu *finish* operasi terakhir pada jadwal awal yang dianggap telah selesai dikerjakan.

3.2.7 Pembentukan Kromosom

Ada dua pendekatan yang diujicobakan sehingga terdapat dua bentuk kromosom untuk permasalahan yang sama yaitu:

1. Penjadwalan kumpulan *job* yang baru datang dengan mengabaikan jadwal lama, pembentukan kromosomnya adalah sebagai berikut:
   a. Semua operasi yang telah dan sedang dikerjakan oleh masing-masing mesin dianggap tidak ada (tidak diperhitungkan).
   b. Kromosom baru beranggotakan kumpulan *job* yang baru datang (belum terjadwalkan)
2. Penjadwalan kumpulan *job* yang baru datang dengan mempertahankan jadwal lama, pembentukan kromosomnya adalah sebagai berikut:
   a. Cek pada Tabel mesin yang menunjukkan posisi *start* dan *finish* tiap operasi.
   b. Semua operasi yang memiliki nilai *start* dibawah waktu kedatangan kumpulan *job* baru dianggap telah selesai dikerjakan, sehingga tidak diikutkan lagi pada proses penjadwalan yang baru.
   c. Waktu *start* mesin pada penjadwalan yang baru adalah waktu *finish* operasi yang terakhir menggunakannya. Bila waktu *finish*nya lebih kecil dari waktu kedatangan *job* baru maka waktu *start* mesin pada penjadwalan yang baru adalah waktu kedatangan *job* baru.

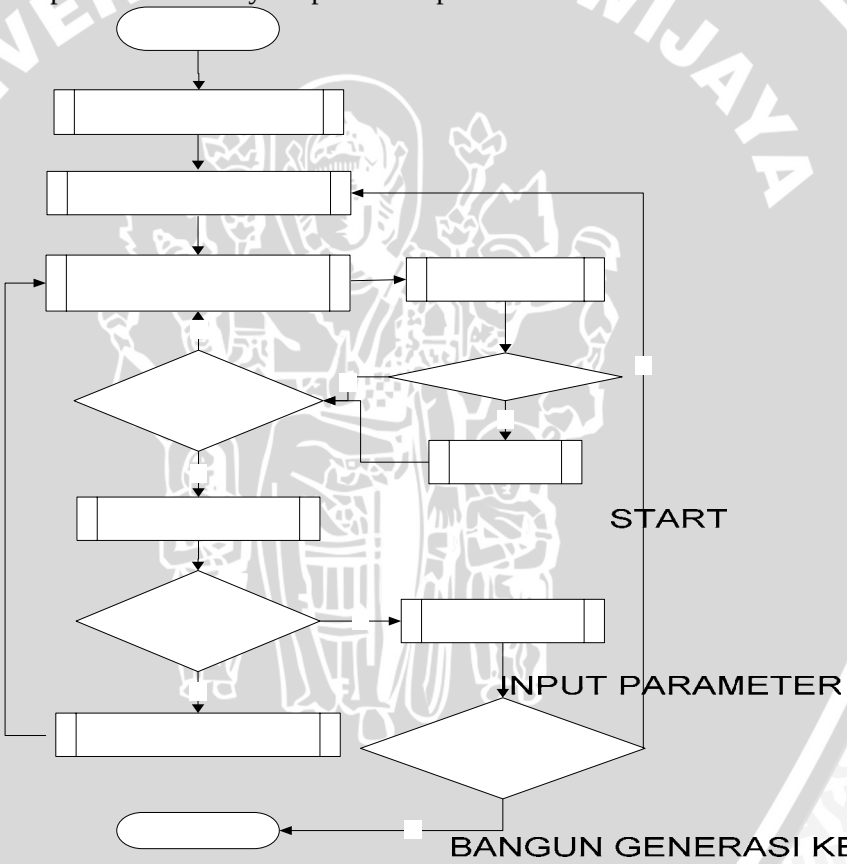
3.2.8 Parameter Genetik

Parameter genetik digunakan untuk mengendalikan operator-operator genetik, parameter yang digunakan adalah: ukuran populasi, jumlah generasi, *probabilitas crossover*, dan *probabilitas* mutasi. Dalam berbagai jurnal, artikel maupun buku teks tidak ditentukan aturan pasti mengenai ukuran parameter genetik tersebut, hanya saja terdapat suatu kesepakatan seperti peluang *crossover* besar dan

peluang mutasi kecil. Karenanya ditetapkan ukuran populasi 20, jumlah generasi 10000, peluang *crossover* 95%, dan peluang mutasi 5%. Jumlah generasi yang besar berarti semakin banyak iterasi yang dilakukan berakibat pada semakin besar daerah solusi yang dieksploitasi. Sedangkan jumlah populasi yang besar berarti memberikan banyak pilihan untuk melakukan *crossover*.

3.2.9 Diagram Alir Algoritma Genetik

Diagram alir dari semua komponen algoritma genetik yang telah disampaikan sebelumnya dapat dilihat pada Gambar 3.5.

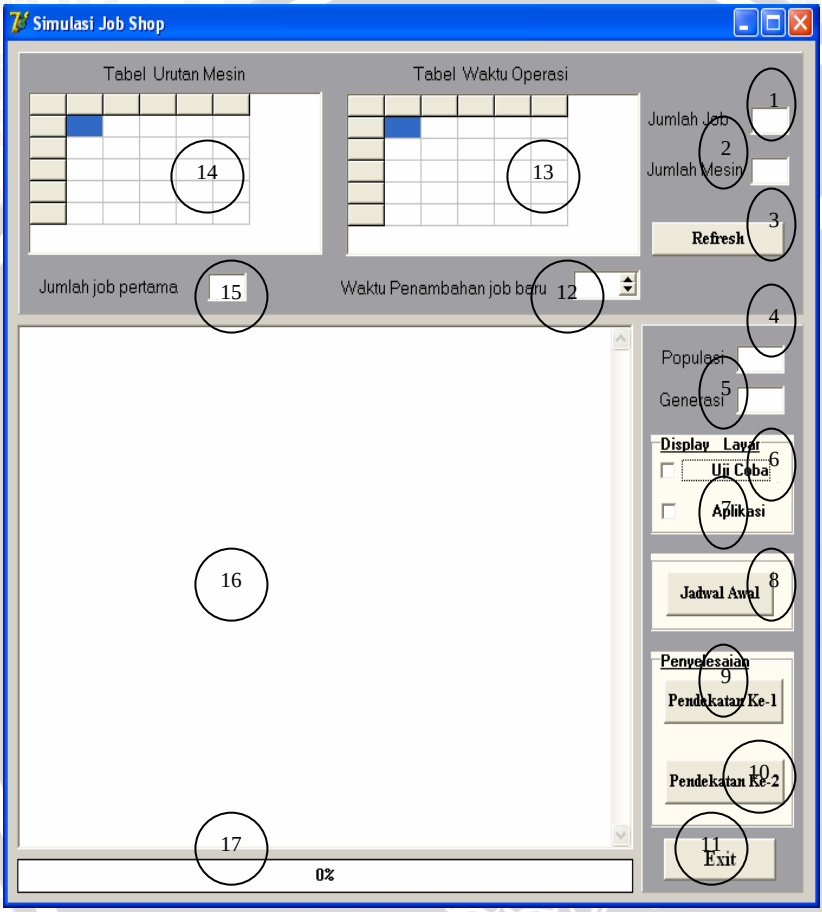


Gambar 3.5 diagram alir algoritma genetik
(Muhammet Yaman , Hamit Saruhan , Faruk Mendi. 2004)

SELEKSI ORANG TUA
ORANG TUA KE-1
ORANG TUA KE-2

3.2.10 Perancangan Antar Muka

Rancangan program untuk mensimulasikan rancangan *job shop* dapat dilihat pada Gambar 3.4.



Gambar 3.6 Rancangan *interface job shop*

Keterangan Gambar 3.4

1. Tempat menginputkan jumlah *job* yang akan dijadwalkan.
2. Tempat menginputkan jumlah mesin keseluruhan yang akan diproses.

3. Tombol *refresh* untuk mendapatkan bilangan yang berbeda pada matrik *job* dan matrik mesin.
4. Tempat menginputkan jumlah populasi untuk tiap generasi.
5. Tempat menginputkan jumlah generasi.
6. Tipe display perhitungan secara keseluruhan.
7. Tipe display hasil dari perhitungan.
8. Tombol jadwal awal yang akan menyusun jadwal berdasarkan jumlah *job* pada nomor 15.
9. Tombol untuk menjalankan cara penjadwalan dengan jalan mengabaikan jadwal yang sudah ada dan menjadwalkan semua *job* dari awal.
10. Tombol Pendekatan kedua yang akan menyusun jadwal berdasarkan operasi yang belum dikerjakan saat jadwal baru datang digabungkan dengan kumpulan *job* baru yang belum terjadwalkan.
11. Tombol untuk keluar dari program.
12. Waktu kedatangan kumpulan *job* kedua.
13. Matrik waktu yang diperlukan masing-masing mesin menyelesaikan operasi.
14. Matrik mesin yang diperlukan masing-masing *job* menyelesikan tiap operasi.
15. Jumlah *job* yang pertama kali dijadwalkan.
16. Memo yang akan menunjukkan langkah demi langkah pencarian solusi.
17. Kotak waktu yang menunjukkan posisi proses pencarian solusi yang sedang berjalan.

3.2.11 Struktur Data

Struktur data yang digunakan dalam penelitian ini ada pada Gambar 3.7

```
Type
  TChromosome = array [1..MAX_STRING_LEN] of integer
  TMatrixCost = array [1..MAX_JOB,1..MAX_JOB] of Real
  TMatrixUrutan = array [1..MAX_JOB,1..MAX_JOB] of Integer
  TStartMesin = array [1..MAX_MESIN,1..MAX_JOB]of Real
  TFinishMesin = array [1..MAX_MESIN,1..MAX_JOB]of Real
  TStartJob = array [1.. MAX_JOB,1.. MAX_MESIN]of Real
  TFinishJob = array [1.. MAX_JOB,1.. MAX_MESIN]of Real
  TJob :array[1..MAX_JOB] of Integer
  TMesin :array[1..MAX_MESIN] of Integer
```

```
  TIndividual = record
       Chrom: TChromosome
       Cost,    Fitness: real
     end;
  TPopulation = array [1..MAX_POPULATIONS] of TIndividual

Var
      CostAwal,FitnesAwal:Real;
      ExcRate,MutRate,waktuTambahJob: Real
      PanjangKromosom:Integer;
      KromosomeAwal : TChromosome
      KromosomeBaru  : TChromosome
      KromosomeTotal : TChromosome
      JOSelesai : TJob
      MOSelesai : TMesin
      StartMesin : TStartMesin
      FinishMesin : TFinishMesin
      StartJob : TStartJob
      FinishJob :  TFinishJob
      OperasiAwal : TChromosome
      MesinAwal : TChromosome
      MesinKeAwal : Tchromosome
      Hasil,Solusi:TIndividual;
      JOperasi : Tjob
      JOperasiSelesai : TJob
      StartMesinAwal : TStartMesin
      FinishMesinAwal : TFinishMesin
      StartJobAwal : TStartJob
      FinishJobAwal :  TFinishJob
      OperasiKe : TChromosome
      MesinKe : TChromosome
      NoMesin : TChromosome
      Job  : TChromosome
      tempOperasi : TJob
      tempMesinAwal   : TMesin
```

Gambar 3.7 Struktur Data

3.3 Perancangan Uji Coba

Tujuan dilakukannya uji coba adalah untuk mengetahui pengaruh waktu penambahan *job* baru dan kompleksitas *job shop* terhadap *makespan* yang dihasilkan oleh kedua pendekatan, dalam menyelesaikan permasalahan *job shop* dinamis.

3.3.1 Data Penelitian

Data yang diperlukan dalam penelitian melingkupi beberapa hal yaitu:

1. Data input
   a. Data urutan mesin dan data waktu pengerjaan tiap operasi, kedua data tersebut didapatkan secara acak (*random*). Data pengerjaan tiap operasi berkisar antara 1 sampai 1,9.
   b. Pada data urutan mesin diberlakukan ketentuan seperti yang disebutkan pada batasan masalah yaitu tiap operasi hanya bisa dikerjakan oleh satu mesin sehingga kemunculan mesin pada *job* yang sama tak boleh berulang.
2. Data proses

   Pada proses pencarian solusi, data urutan mesin dan data waktu pengerjaan tiap operasi digunakan untuk menentukan nilai *makespan*. Proses pencarian solusi menggunakan algoritma genetik.
3. Data output
   a. Data Output yang digunakan adalah *makespan*.
   b. Data *makespan* optimum tiap kondisi (jumlah *job* dan waktu penambahan *job*) ditulis dalam bentuk tabel.
   b. Nilai *makespan* optimum pada tiap kondisi digambarkan dalam bentuk grafik (contoh Gambar 3.6) untuk mempermudah mengetahui pergerakan data *makespan*.

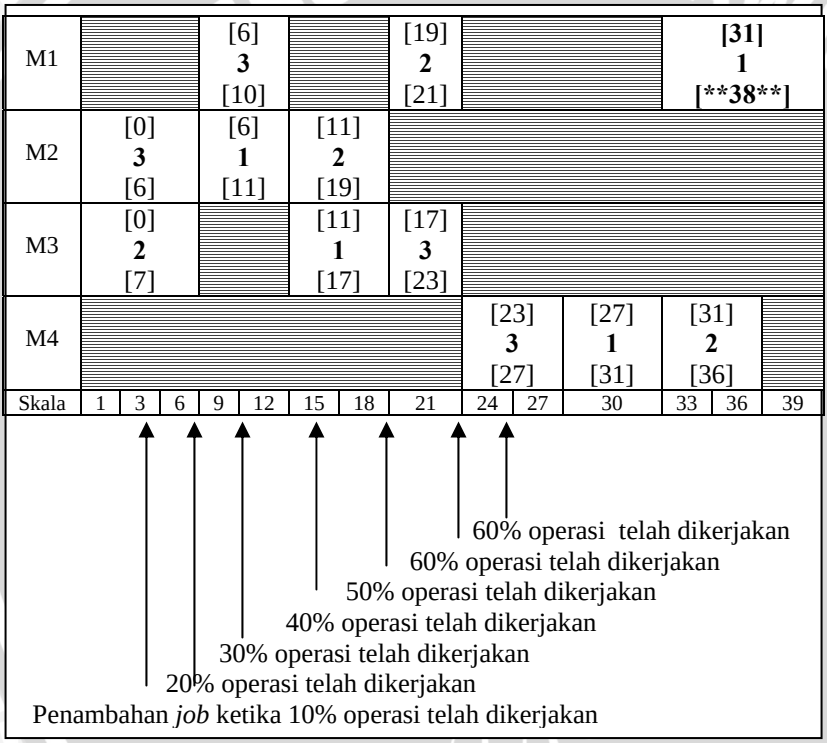
3.3.2 Skenario Pengujian

Uji coba dilakukan dengan ketentuan sebagai berikut :

a. Banyaknya *job* bertingkat misalkan dimulai dari 16,20 dan 32 tujuannya untuk mengetahui pengaruh kompleksitas *job shop* terhadap *makespan* yang dihasilkan.
b. Jumlah operasi pada tiap *job* sama dengan jumlah mesin yang mengerjakannya, misalnya untuk jumlah total *job* 24 diproses oleh 10 mesin maka tiap *job* memiliki 10 operasi dengan kata lain ada 240 operasi. Hal ini diberlakukan untuk menjamin bahwa setiap penambahan jumlah *job* atau mesin berarti jumlah operasi (kompleksitas *job shop*) juga meningkat.
c. Jumlah *Job* pertama yang diproses bertingkat yaitu 1/4 dan 3/4 dari total *job* yang akan di kerjakan. Misalkan untuk total *job* 20 maka *Job* pertamanya adalah 5 dan 15. Hal ini dilakukan untuk

mengetahui pengaruh jumlah operasi yang diabaikan atau diteruskan pada pendekatan 1 dan pendekatan 2 terhadap *makespan* yang dihasilkan.

d. Waktu penambahan *job* baru diletakkan pada 7 tempat disesuaikan pada *makespan* jadwal yang terbentuk dari kumpulan *job* pertama yaitu sebesar 10%, 20%, 30%, 40%, 50%, 60% dan 70% dari *makespan* jadwal awal. Pada Gambar 3.8 dapat dilihat dengan menggunakan *makespan* 38 maka terdapat waktu penambahan *job* baru :



Gambar 3.8 Ilustrasi tempat penambahan *job* baru

Dengan menggunakan *makespan* 38 waktu penambahan *job* baru untuk 10% adalah 3.8 , kemudian berturut-turut 20% ,30%, 40%, 50%, 60%, 70% adalah 7.6, 11.4, 15.2, 19, 22.8, dan 26.6.

Keterangan Gambar 3.8

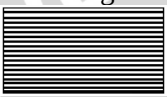
1. keterangan proses penjadwalan

[31] ------- waktu *start* mesin
2 ------- nama *job*
[36] ------- waktu *finish* mesin

2. keterangan penentuan nilai *makespan*

[31] ------- waktu *start* mesin
1 ------- nama *job*
[38**]** ------- waktu *finish* mesin (***MAKESPAN***)

3. keterangan waktu mengangur mesin (*idle time*)

 → waktu menganggur

e. Tiap-tiap kondisi percobaan diulang sebanyak 5 kali, dan diambil nilai terbaik. Hal ini dilakukan karena algoritma yang digunakan adalah algoritma genetik (heuristik) yang memungkinkan melewatkan solusi yang terbaik.
f. Untuk pendekatan pertama nilai *makespan* pada tiap kondisi didapatkan dengan menambahkan *makespan* dari jadwal yang didapat (nilai optimum dari 5 kali percobaan) dengan waktu penambahan *job* baru.
g. Hasil dari perhitungan *fitness* pada tiap-tiap kondisi digambarkan dengan grafik seperti pada Gambar 3.9.

Gambar 3.9 Rancangan grafik

3.3.3 Contoh Penghitungan Manual

Pada contoh penghitungan manual ini terdapat dua proses utama yang dikerjakan yaitu pembentukan jadwal awal berdasarkan kumpulan *job* yang datang pertama kali kemudian proses yang kedua adalah penyusunan jadwal baru setelah ada penambahan beberapa *job* baru. Detail prosesnya akan diuraikan sebagai berikut :

1. Pembentukan jadwal awal berdasarkan kumpulan *job* yang datang pertama kali.

Untuk melakukan penghitungan manual digunakan data-data pada Tabel 3.1, yaitu terdapat 3 *job* dengan masing-masing 4 buah operasi. Untuk *job* 1 operasi pertama dikerjakan di mesin 2, operasi kedua dikerjakan di mesin 3, operasi ketiga dikerjakan di mesin 4, dan operasi keempat dikerjakan di mesin 1, begitu pula pada *job* 2 dan *job* 3.

Tabel 3.1 *Job* beserta mesin yang dibutuhkan

JOB	**MESIN**			
1	2	3	4	1
2	3	2	1	4
3	2	1	3	4

Tabel 3.2 *Job* beserta waktu pengerjaan tiap operasi

Job	**Waktu**			
1	5	6	4	7
2	7	8	2	5
3	6	4	6	4

Tabel 3.2 memberikan informasi mengenai lamanya waktu proses tiap operasi pada masing-masing mesin dalam satuan waktu tertentu. Untuk *job* 1 operasi pertama dikerjakan di mesin 2 selama 5 satuan waktu, operasi kedua dikerjakan di mesin 3 selama 6 satuan waktu, operasi ketiga dikerjakan di mesin 4 selama 4 satuan waktu, dan operasi keempat dikerjakan di mesin 1 selama 7 satuan waktu, begitu pula pada *job* 2 dan *job* 3. Dari tabel 3.1 tersebut didapatkan inisialisasi kromosomnya sebagai berikut:

Inisialisasi kromosom :

1	1	1	1	2	2	2	2	3	3	3	3

Contoh kromosom lainnya :

1	2	3	2	3	1	2	3	1	1	3	2
3	2	1	2	1	1	2	3	3	1	3	2
3	1	2	3	2	2	1	3	1	3	1	2
2	3	1	2	1	3	2	3	3	1	2	1

Job 1 operasi ke-1

Job 3 operasi ke-1

Job 2 operasi ke-1 dan seterusnya.

Proses pencarian jadwal yang optimal ini termasuk kedalam kasus minimasi sehingga bila digunakan variabel **h** sebagai *makespan* maka nilai fitnesnya adalah 1/h. Dengan menggunakan nilai *fitness* 83, 67, 50, 45, dan 38 penentuan peluang terpilihnya kromosom menjadi *parent* (orang tua) menggunakan metode *roulette wheel* dapat dilihat pada Tabel 3.3.

Tabel 3.3 Ilustrasi *roulette wheel*

individu	***Fitness*** **(1/h)**	**prob**	**prob cumulatif**	**prob cum *100**	**Range**	**Selisih Range**
1	83	0.29	0.29	29	1—29	29
2	67	0.23	0.52	52	30—52	22
3	50	0.17	0.69	69	53—69	16
4	45	0.15	0.84	84	60—84	14
5	38	0.13	0.97	97	85—97	12
∑ ***Fitness***	283					

Setelah tabel terbentuk maka untuk mendapatkan *parent*, tinggal membangkitkan bilangan random antara 1-97. Berdasarkan selisih *range* pada Tabel 3.3 dapat dilihat bahwa individu nomer satu memiliki peluang muncul lebih besar dibandingkan individu yang lain, sesuai dengan nilai *fitness*nya yang paling besar, contoh operasi *crossover* pada Gambar 3.10.

Parent1	1	3	2	3	2	1	2	3	2	3	1	1
Offspring	1	3	2	3	1	2	2	1	3	2	3	1
Parent2	2	3	1	3	1	2	2	1	3	3	1	2

Gambar 3.10 Ilustrasi *crossover*

Pada *crossover* ini memperhatikan jumlah operasi masing-masing *job* sehingga bila jumlah operasi sudah lengkap maka operasi selanjutnya digantikan dengan operasi yang belum lengkap jumlahnya. Sedangkan operasi mutasi yang diuji cobakan seperti pada Gambar 3.11.

Sebelum mutasi	2	3	1	3	1	2	2	1	3	3	1	2
Sesudah mutasi	2	3	1	1	2	2	1	3	3	1	2	3

Gambar 3.11 Ilustrasi mutasi

Dengan menggunakan data pada Tabel 3.1 dan Tabel 3.2 maka penghitungan nilai *fitnes*nya adalah sebagai berikut:
Sebagai contoh kromosom yang akan dihitung nilai *fitnes*nya adalah 2-3-1-2-1-3-2-3-3-1-2-1 maka langkah 1-3 dapat digambarkan pada Gambar 3.12.

Job	2	3	1	2	1	3	2	3	3	1	2	1
Operasi ke-	1	1	1	2	2	2	3	3	4	3	4	4
NoMesin	3	2	2	2	3	1	1	3	4	4	4	1

Gambar 3.12 Ilustrasi langkah 1-3

Pada Gambar 3.7 nilai dari ***job*** adalah nilai tiap gen dari kromosom yang sedang dihitung nilai *fitnes*snya. Sedangkan nilai dari **operasi ke-** merupakan keterangan operasi ke berapa yang sedang dijadwalkan, nilai tersebut didapatkan dari berapa kali *job* tersebut muncul di kromosom. **No Mesin** merupakan keterangan di mesin keberapakah operasi tersebut akan dikerjakan, nilai ini didapatkan dengan mencocokkan nilai ***job*** dan **operasi ke-** pada Tabel 3.1.
Langkah 4-6 untuk operasi pertama ada pada Gambar 3.13:

Mesin 1	S			
	F			
Mesin 2	S	0	6	
	F	6	11	
Mesin 3	S	0		
	F	7		
Mesin 4	S			
	F			

Gambar 3.13 Waktu *start* dan *finish* mesin untuk

Dengan menggunakan kromosom yang sama (2-3-1-2-1-3-2-3-3-1-2-1) berarti operasi pertama yang dijadwalkan adalah *job* 2 di mesin 3, *job* 3di mesin 2 kemudian *job* 1 di mesin 2. Waktu proses masing-masing operasi didapatkan dengan mencocokkan nama *job* dan operasi ke- pada Tabel 3.2. Untuk operasi pertama ini *job* 2 operasi 1 waktu prosesnya =7, *job* 3 operasi 1 waktu prosesnya=6, dan *job* 1 operasi 1 waktu prosesnya=5. penjadwalan bersifat *non preemtif* sehingga jika ada operasi yang membutuhkan mesin yang sama untuk diproses maka operasi tersebut baru bisa diproses setelah operasi sebelumnya selesai dikerjakan. Pada Gambar 3.8 terlihat *job* 3 operasi 1 dan *job* 1 operasi 1 sama –sama menggunakan mesin 2 maka penjadwalannya sesuai urutan pada kromosom, yaitu *job* 3 dulu kemudian baru *job* 1. Untuk *job* 2 karena pada mesin 3 tidak ada yang menggunakan maka bisa langsung diproses tanpa menunggu *job* yang lain.

Waktu *start* dan *finish* tiap *job* dapat dilihat pada Gambar 3.9. *job* 1 untuk operasi 1 waktu *start* nya (S) 6 bukan 0 karena *job* tersebut dalam penjadwalannya harus menunggu terselesaikannya *job* 3 diproses. Sedangkan *job* 2 dan *job* 3 waktu *start* nya (S) adalah 0 karena tidak menunggu operasi lain selesai dikerjakan seperti pada Gambar 3.13, sedangkan pada Gambar 3.14 dan Gambar 3.16 adalah proses penghitungan nilai *makespan* semua operasi .

Langkah ke-7 adalah membandingkan nilai *finish* di tiap mesin pada operasi terakhir yang di kerjakan, nilai yang paling besar itulah nilai *fitness*nya (*makespan*).

Mesin 1 = 38, Mesin 2 = 19, Mesin 3 = 23 dan Mesin 4 = 36, sehingga didapatkan nilai *fitness*nya sebesar 38.

Untuk memudahkan dalam memahami langkah-demi langkah pada Gambar 3.17 dilakukan penghitungan *makespan* dengan menggunakan grafik (*gant chart*).

Job 1	S	6			
	F	11			
Job 2	S	0			
	F	7			
Job 3	S	0			
	F	6			

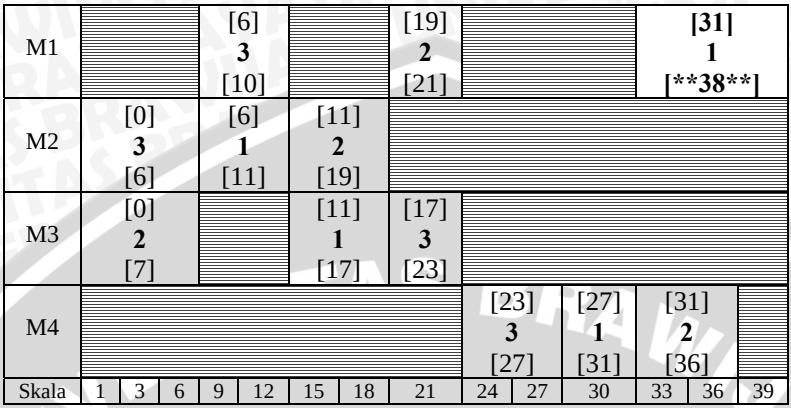
Gambar 3.14 Waktu *Start* dan *Finish Job* langkah 4-6 operasi pertama

Mesin 1	S	6	19	31
	F	10	21	38
Mesin 2	S	0	6	11
	F	5	11	19
Mesin 3	S	0	11	17
	F	7	17	23
Mesin 4	S	23	27	31
	F	27	31	36

Gambar 3.15 Waktu *start* dan *finish* mesin langkah 4-6 semua operasi

Job 1	S	6	11	27	31
	F	11	17	31	38
Job 2	S	0	11	19	31
	F	7	19	21	36
Job 3	S	0	6	17	23
	F	6	10	23	27

Gambar 3.16 Waktu *Start* dan *Finish*h *Job* langkah 4-6 semua operasi



Gambar 3.17 Ilustrasi penghitungan *fitnes*

2. Penyusunan jadwal baru setelah ada penambahan beberapa *job* baru.

Untuk pembentukan kromosom pada penjadwalan dengan mempertahankan sebagian jadwal yang ada, digunakan sebagai contoh data pada Gambar 3.16 dengan waktu kedatangan adalah 40% dari *makespan* (40% dari 38 = 15.2) maka operasi yang dianggap telah selesai dapat dilihat pada Gambar 3.18 berikut:

Job 1	S	6	11		
	F	11	17		
Job 2	S	0	11		
	F	7	19		
Job 3	S	0	6		
	F	6	10		

Gambar 3.18 Operasi-operasi yang dianggap selesai dikerjakan

Sedangkan untuk pembentukan kromosom pada penjadwalan dengan mengabaikan jadwal yang ada (menjadwalkan dari awal), operasi-operasi yang sudah dan sedang diproses dianggap tidak ada. Sehingga waktu *start* pada tiap mesin untuk penjadwalan yang baru (*job* lama + *job* yang baru datang) adalah 15 (waktu kedatangan *job* baru). Sedangkan berdasarkan data pada Gambar 3.15 untuk penjadwalan dengan mempertahankan sebagiang jadwal yang ada waktu *start*nya dapat dilihat pada Gambar 3.19.

Mesin 1	S	6	15	
	F	10		
Mesin 2	S	0	6	11
	F	5	11	19
Mesin 3	S	0	11	17
	F	7	17	
Mesin 4	S	15		
	F			

Gambar 3.19 Nilai *start* tiap mesin pada jadwal yang baru

Caranya adalah mencari di tiap mesin yang memiliki nilai *start* terbesar yang mendekati 15 (waktu kedatangan *job* baru) tetapi tak sampai melebihi 15. Kemudian nilai *start* yang baru adalah nilai *finish* dari pasangan *start* yang terakhir kali menggunakan mesin tersebut. Mesin 1 waktu *start*nya 10 karena pada mesin 1 operasi terakhir yang menggunakannya adalah operasi ke-2 pada *job* 3 dengan waktu *finish* 10, hal yang sama diberlakukan pada mesin yang lain (mesin 2, mesin 3 dan mesin 4). Jika operasi pertama sudah memiliki waktu *start* diatas 15 maka waktu *start* yang baru adalah waktu kedatangan *job* baru yaitu 15 (contoh mesin 1 dan 4).

3. Pembentukan kromosom jika ada *job* yang baru datang

a. Untuk penjadwalan dangan mengabaikan operasi yang sudah dan sedang berjalan (penjadwalan dari awal).

Dengan tetap menggunakan data pada tabel 3.1 dan Tabel 3.2 dikondisikan *job* yang baru datang adalah *job* 4 dan *job* 5 seperti terlihat pada Tabel 3.4 dan Tabel 3.5 dengan waktu kedatangan 40% dari *makespan* jadwal yang lama (40% dari 38 =15).

Tabel 3.4 urutan mesin *job* baru

JOB	**MESIN**			
4	4	3	2	1
5	2	4	1	3

Tabel 3.5 waktu operasi *job* baru

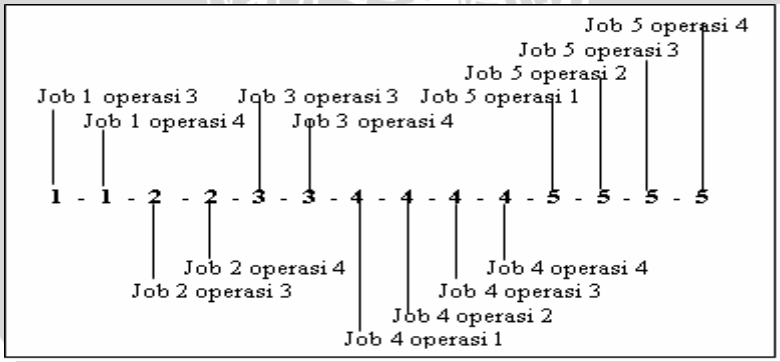
JOB	Waktu			
4	7	5	2	8
5	5	6	4	9

Berdasarkan data Tabel 3.1 dan Tabel 3.4 maka bentuk inisialisasi kromosomnya adalah:

4	4	4	4	5	5	5	5

b. Untuk penjadwalan dangan memperhatikan operasi yang sudah dan sedang berjalan.

Dengan tetap menggunakan data pada tabel 3.1 dan Tabel 3.2 dikondisikan *job* yang baru datang adalah *job* 4 dan *job* 5 seperti terlihat pada Tabel 3.4 dan Tabel 3.5 dengan waktu kedatangan 40% dari *makespan* jadwal yang lama (40% dari 38 =15). Karena memperhatikan operasi yang sedang dan telah dikerjakan maka harus memperhatikan Gambar 3.18. Dari Gambar 3.19 diketahui bahwa *job* 1 operasi yang telah dan sedang dikerjakan adalah operasi 1 dan 2, begitu pula yang terjadi pada *job* 2 dan *job* 3. Sehingga operasi pertama dan kedua pada *job* 1, 2 dan 3 tidak lagi diikutkan dalam penjadwalan yang baru (*job* lama +*job* baru), sehingga inisialisasi kromosomnya seperti pada Gambar 3.20.



Gambar 3.20 Arti dari inisialisasi kromosom

Penghitungan nilai *makespan* untuk jadwal baru prosesnya sama seperti menghitung *makespan* pada jadwal awal hanya saja

harus memperhatikan operasi yang telah selesai dikerjakan, dengan kata lain tinggal melanjutkan mengerjakan operasi yang belum selesai dikerjakan.Misalkan kromosom yang akan dihitung adalah

1	**2**	**2**	**4**	**3**	**3**	**4**	**1**	**5**	**5**	**4**	**4**	**5**	**5**

maka dengan menggunakan Gambar 3.18 dan Gambar 3.19 sebagai data awal, penghitungan *makespan*nya ada pada Gambar 21 dan Gambar 3.22.

Mesin 1	S	**6**	19	21	40	48
	F	**10**	21	28	48	52
Mesin 2	S	**0**	**6**	**11**	19	38
	F	**5**	**11**	**19**	24	40
Mesin 3	S	**0**	**11**	17	33	52
	F	**7**	**17**	23	38	**61**
Mesin 4	S	17	21	26	33	37
	F	21	26	33	37	43

Gambar 3.21 Waktu *start* dan *finish* mesin untuk jadwal baru

Job 1	S	**6**	**11**	17	21
	F	**11**	**17**	21	28
Job 2	S	**0**	**11**	19	21
	F	**7**	**19**	21	26
Job 3	S	**0**	**6**	17	33
	F	**6**	**10**	23	37
Job 4	S	26	33	38	40
	F	33	38	40	48
Job 5	S	19	37	48	52
	F	24	43	52	61

Gambar 3.22 Waktu *start* dan *finish Job* untuk jadwal baru

Pada Gambar 3.21 dan Gambar 3.22 angka yang dicetak tebal adalah perhitungan dari jadwal awal, sedangkan yang ditandai dengan bintang-bintang (**61**) adalah *makespan* jadwal baru.

BAB IV
HASIL DAN PEMBAHASAN

Pada bab ini akan dibahas mengenai implementasi dari representasi rancangan ke bahasa pemrograman yang dimengerti oleh komputer. Selanjutnya hasil implementasi yang dihasilkan oleh perangkat lunak, akan dievaluasi sebagai bahan pengambil kesimpulan .

4.1 Lingkungan Implementasi

Lingkungan implementasi yang akan dijelaskan dalam subbab ini adalah lingkungan implementasi perangkat keras dan perangkat lunak.

4.1.1 Lingkungan perangkat keras

Perangkat keras yang digunakan dalam pengembangan sistem penjadwalan *Job Shop* ini adalah sebagai berikut :

1. Prosesor Intel(R) Pentium(R) 2.4 GHz
2. Memori 256 MB
3. Harddisk dengan kapasitas 40 GB
4. Monitor 17 ”
5. Keyboard
6. Mouse

4.1.2 Lingkungan perangkat lunak

Perangkat lunak yang digunakan dalam pengembangan sistem penjadwalan *Job Shop* ini adalah :

1. Sistem Operasi Windows XP SP 1
2. Borland Delphi 7

4.2 Implementasi Program

Berdasarkan analisis dan perancangan proses yang terdapat pada subbab 3.2, maka pada subbab ini akan dijelaskan implementasi proses-proses tersebut.

4.2.1 Pembentukan Data Penelitian

Tahap awal dari penjadwalan *JobShop* ini adalah mengambil data *input* jumlah mesin dan *job*. Setelah itu dilakukan pembangkitan bilangan acak untuk urutan mesin pada tiap *job* dan waktu pengerjaan tiap-tiap operasi. Listing program untuk melakukan generate data ada pada *SourceCode* 4.1.

```
SGMesin.RowCount:=strtoint(Edit job.text)+1;
  SGMesin.ColCount:=strtoint(Editmesin.Text)+1;
  //inisisalisasi posisi mesin terurut sebelum diacak
   for baris:=1 to nJob+1 do
     begin
        for kolom:=1 to nMesin do
           Mesin[kolom,baris]:=kolom;
     end;
  //inisisalisasi posisi mesin Temp
  for baris:=1 to 2 do
     begin
       for kolom:=1 to 2 do
           MesinTemp[kolom,baris]:=kolom;
     end;
```

SourceCode 4.1 proses pengisian urutan mesin tiap *job*

Pada SourceCode 4.1, akan dihasilkan *string grid* dengan jumlah kolom dan baris yang sesuai dengan jumlah *job* dan mesin yang di*input*kan, dimana kolom mengindikasikan jumlah mesin dan baris mengindikasikan jumlah *job*. *Array* `Mesin[j,baris]` dan array `MesinTemp` diberi nilai awal urut dari 1 sampai jumlah mesin. Array `MesinTemp` digunakan untuk melakukan pengacakan urutan mesin pada array `Mesin`, proses pengacakannya ada pada SourceCode 4.2.

```
  // pengacakan posisi mesin tiap job
  for baris:=1 to nJob+1 do
    begin
      for kolom:=1 downto nMesin do
        i:=0;j:=0;
        repeat
          i:=(random(nMesin))+1;
          j:=(random(nMesin))+1;
        until i<>j;
        MesinTemp[1,1] :=Mesin[j,baris];
        Mesin[j,baris] :=Mesin[i,baris];
        Mesin[i,baris] :=MesinTemp[1,1];
   end;
```

SourceCode 4.2 proses pengacakan urutan mesin

Pada SorceCode 4.2 dilakukan proses penukaran urutan mesin sebanyak jumlah mesin. Urutan mesin yang sudah diacak disimpan kembali di array `Mesin[j,i]`. Selanjutnya urutan mesin yang sudah teracak ditampilkan kembali di *string grid* seperti pada SourceCode 4.3.

```
      //memasukkan nilai ke tabel mesin (SgMesin)
  for i:=0 to SGMesin.RowCount+1 do
    begin
      for j:=0 to SGMesin.ColCount+1 do
         if (i=0) then
            SGMesin.Cells[j,i] :='m O'+IntToStr(J)
         else if (j=0) then
            SGMesin.Cells[j,i] :='J'+IntToStr(I)
         else
           SGMesin.Cells[j,i] :=IntToStr(Mesin[j,i]);
           SGMesin.Cells[0,0]:='';
   end;
```

SourceCode 4.3. proses menampilkan urutan mesin

Untuk waktu pengerjaan tiap operasi didapatkan dengan membangkitkan bilangan acak 1 sampai 10 dibagi 10 dan ditambah dengan 1 seperti pada SourceCode 4.4. Waktu operasi kemudian ditampilkan di *string grid* waktu (`SGWaktu`).

```
//mengisi tabel waktu
  SGWaktu.RowCount:=strtoint(Edit job.text)+1;
  SGWaktu.ColCount :=strtoint(Editmesin.Text)+1;
  for i:=0 to SGWaktu.RowCount+1 do
    begin
      for j:=0 to SGWaktu.ColCount+1 do
                    if (i=0) then
           SGWaktu.Cells[j,i] :='t O'+IntToStr(J)
       else if (j=0) then
           SGWaktu.Cells[j,i] :='J'+IntToStr(I)
       else
            SGWaktu.Cells[j,i]:= :=FloatToStr
                                  (1+(Random(10)/10));
           SGWaktu.Cells[0,0]:='';
   end;
```

SourceCode 4.4 proses mengisi waktu pengerjaan tiap *job*

4.2.2 Inisialisasi Individu

Pembentukan individu pertama dilakukan dengan menyusun masing-masing *job* sebanyak operasi yang dimiliki, untuk

pembentukan jadwal awal, tiap *job* memililki operasi sebanyak jumlah mesin sedangkan untuk pembentukan jadwal setelah ada penambahan *job* baru maka jumlah operasinya disesuaikan dengan jumlah operasi yang belum dikerjakan, implementasinya pada SourceCode 4.3.

```
var i,j,k:integer;
begin
  k:=1;
 for i:=1 to nJob do
    for j:=1 to JOperasi[i] do
    begin
     IndInitial.Chrom[k] := i;
    inc(k);
    end;
end;
```

SourceCode 4.5 Inisialisasi individu

Pada SourceCode 4.5 terdapat array `JOperasi[i]`, array tersebut berisikan jumlah operasi pada masing-masing *job*. Sedangkan `IndInitial.Chrom[k]` adalah array yang menyimpan individu pertama sebagai dasar pencarian jadwal.

4.2.3 Proses Crossover

Pada SourceCode 4.6 langkah pertama *crossover* adalah mencari dua titik potong (`cut1 dan cut2`).

```
var
  i,cut1,cut2:integer;
  LPar1,LPar2:TList;
begin
   lPar1.Init;
   LPar2.Init;
   for i:=1 to chromLen do
    begin
      LPar1.Insert(Par1.Chrom[i]);
      LPar2.Insert (Par2.Chrom[i]);
    end;
   cut1 := Rnd (chromLen);
   repeat //  cari dua titik potong
     cut2 := Rnd (chromlen);
   until (cut1<>cut2);
   if (cut1 > cut2) then
      SwapInt(cut1,cut2);
```

SourceCode 4.6 proses mencari dua titik potong

```
for i:=1 to chromlen do
 begin
   if (i<=cut1) then
     begin  // copy from Parent1
       Child.Chrom[i] := Par1.Chrom[i];
       LPar1.DeleteElement(Child.Chrom[i]);
       LPar2.DeleteElement (Child.Chrom[i]);
     end
  else if (i>cut1)and (i<=cut2) then
     begin   // copy from Parent2
   Child.Chrom[i] := LPar2.A[i-cut1];
       LPar1.DeleteElement(Child.Chrom[i]);
     end
  else
     begin  // copy from Parent1
       Child.Chrom[i] := LPar1.A[i-cut2];
     end;
 end;
```

SourceCode 4.7 Proses *Crossover*

SourceCode 4.7 menjelaskan pada kromosom anak (`Child.Chrom[i]`) gen yang ada dibawah titik potong pertama didapat dari *parent* 1, gen yang ada di antar titik potong pertama dan kedua didapatkan dari *parent* 2 sedangkan gen yang berada diatas titik potong kedua didapatkan dari *parent* 1. Proses perbaikan *crossover* dilakukan dengan menghilangkan gen yang sudah di*copy*kan ke gen anak. Jadi setiap dilakukan pengisian gen anak baik berasal dari *parent* 1 maupun *parent* 2 selalu diikuti dengan penghapusan gen yang serupa pada kedua *parent* (`LPar1.DeleteElement (Child.Chrom[i])`).

4.2.4 Proses Mutasi

Mutasi yang dipakai dalam penelitian ini adalah pindah sisip *(Shif mutation*). Mutasi yang dimaksud adalah dengan cara memilih dua gen secara acak kemudian gen pertama tempatnya ditukar jadi menempati gen kedua, sedangkan gen antara gen pertama dan kedua digeser, implementasinya ada pada SourceCode 4.8. Mula-mula cari duabuah titik potong (`site1,site2`). Kemudian dilakukan proses penyisipan dengan cara menukar posisi pertama menjadi posisi titik potong kedua. Posisi gen yang ditinggalkan ditambah 1, sehingga mengisi tempat didepannya (`Ind.Chrom[a] := Ind.Chrom[a+1];`).

```
procedure TGeneticJobShop.Mutation (var
Ind:TIndividual);
var
  a,b,site1,site2:integer;
begin
  site1 := Rnd(chromLen);
  repeat
    site2 := Rnd(chromLen);
  until site1<>site2;
  if site1>site2 then
    begin
      SwapInt (site1, site2);
    end;
  b := Ind.Chrom[site1];
  for a:=site1 to site2-1 do
    begin
     Ind.Chrom[a] := Ind.Chrom[a+1];
    end;
  Ind.Chrom[site2] := b;

end;
```

SourceCode 4.8 Proses Mutasi

4.2.5 Penghitungan *Makespan*

Untuk melakukan penghitungan *makespan* ada beberapa langkah dalam implementasi yaitu *inisialisasi*, *input* nilai dan penghitunan *makespan*. *Inisialisasi* dilakukan untuk menyiapkan data penghitungan. *Input* nilai dilakukan untuk mengisi data-data penghitungan, sedangkan penghitungan *makespan* dilakukan untuk menghitung waktu *start* dan *finish* tiap-tiap operasi. Untuk penghitungan *makespan* dibagi lagi dalam 4 kondisi yaitu

1. Jika digunakan pertama kali mesin memproses operasi pertama pada salah satu *job*.
2. Jika mesin digunakan pertama kali tetapi yang dikerjakan bukanlah operasi pertama.
3. Jika operasi pertama pada *job* yang dikerjakan tapi mesin sudah pernah mengerjakan operasi yang lain sebelumnya.
4. mesin sudah pernah digunakan sebelumnya dan operasi yang sedang dikerjakan bukanlah operasi pertama.

Pada SourceCode 4.9 melakukan proses inisialisasi, tahap inisialisasi dilakukan dengan mengisi nilai untuk array `tempOperasi[i]` dan `tempMesin[i]`. Kedua array tersebut berguna untuk menentukan operasi ke berapa yang akan dijadwalkan.

```
//*************inisialisasi
  if (Status=1) then
    begin
      for i:=1 to nJob do
        begin
          tempOperasi[i]:=0;
        end;
      for i:=1 to nMesin do
        begin
          tempMesin[i]:=tempMesinAwal[i];
        end;
     end
   else
      for i:=1 to nJob do
        begin
           tempOperasi[i]:=nMesin-JOperasi[i];
        end;
      for i:=1 to nMesin do
        begin
          tempMesin[i]:=tempMesinAwal[i];
        end;
```

SourceCode 4.9 Tahap inisialisasi

Status yang dimaksudkan di *SourceCode 4.9* adalah jika jadwal yang sedang dihitung mengalami penambahan *job* baru maka statusnya adalah 2 sedangkan yang tak mengalami penambahan *job* baru statusnya adalah 1.

```
//****************bagian input nilai
  for i:=1 to chromLen do //input Urutan operasi
   begin
      Job[i]:=ind.Chrom[i];
   end;
//input operasi ke- berapa dalam job tsb
  for i:=1 to chromLen do      begin
     tempOperasi[Job[i]]:=tempOperasi[Job[i]]+1;
     OperasiKe[i]:=tempOperasi[Job[i]];
   end;
  for i:=1 to chromlen do  // input mesin yang dipakai
   begin
      temp:= MUrutan[OperasiKe[i],Job[i]];
      NoMesin[i]:=temp;
   end;
//input mesin dipakai ke- berapa
  for i:=1 to chromLen do      begin
      tempMesin[NoMesin[i]]:=tempMesin[NoMesin[i]]+1;
      MesinKe[i]:=tempMesin[NoMesin[i]];
   end;
```

SourceCode 4.10 Tahap input nilai

Setelah proses inisialisasi pada SourceCode 4.9, pada SourceCode 4.10 melakukan proses input nilai. Dimana data diatur supaya dikenali dan dapat diolah (dilakukan peghitungan *makespan*). Array `Job[i]` digunakan untuk menyimpan data kromosom. Kemudian dilakukan pengecekan tiap index (*gen*) untuk menentukan operasi keberapa, dengan bantuan array `tempOperasi[i]` dan hasilnya disimpan dalan array `OperasiKe[i]`. Array `tempOperasi[i]` dan `OperasiKe[i]` digunakan bersama-sama untuk menentukan mesin mana yang akan mengerjakan operasi tersebut pada `MUrutan[OperasiKe[i],Job[i]]` dan hasilnya disimpan dalam array `NoMesin[i]`. Array `MUrutan[OperasiKe[i],Job[i]]` dan `NoMesin[i]` digunakan bersama-sama untuk menentukan urutan pengerjaan operasi pada masing-masing mesin dengan bantuan array `tempMesin[NoMesin[i]]` dan hasilnya disimpan dalam array `MesinKe[i]`.

```
//copy penghitungan  Job jadwal awal
for a:=1 to nJob do
    begin
      for b:=1 to nMesin do
        begin
          StartJob[a,b]:=StartJobAwal[a,b];
          FinishJob[a,b]:=FinishJobAwal[a,b];
        end;
    end;
// copy penghitungan  Mesin jadwal awal
  for a:=1 to nMesin do
    begin
      for b:=1 to nJob do
        begin
          StartMesin[a,b]:=StartMesinAwal[a,b];
          FinishMesin[a,b]:=FinishMesinAwal[a,b];
        end;
    end;
```

SourceCode 4.11 proses pengkopian penghitungan

Pada SourceCode 4.11 dilakukan proses pengkopian data penghitungan *makespan*. Pengkopian ini dilakukan sebagai alat penghitungan *makespan* untuk jadwal yang mengalamai penambahan *job* baru. Array `StartJobAwal[a,b]` dan `FinishJobAwal[a,b]` adalah array yang menyimpan data penghitungan *start* dan *finish job* pada jadwal Awal sedangkan array `StartMesinAwal[a,b]` dan `FinishMesinAwal[a,b]` adalah array yang menyimpan data penghitungan *start* dan *finish* mesin pada jadwal awal.

```
//operasi pertama job itu dan mesin digunakan
  pertama kali
    if (MesinKe[i]=1)and (OperasiKe[i]=1) then
      begin
        if (Status=2) then
          begin
             StartMesin[NoMesin[i],MesinKe[i]]:=
                                        WaktuTambahJob;
          end
        else
          begin
            StartMesin[NoMesin[i],MesinKe[i]]:=0;
          end;
          FinishMesin[NoMesin[i],MesinKe[i]]:=
                            MCost[OperasiKe[i],Job[i]];
          StartJob[Job[i],OperasiKe[i]]:=
                     StartMesin[NoMesin[i],MesinKe[i]];
          FinishJob[Job[i],OperasiKe[i]]:=
                    FinishMesin[NoMesin[i],MesinKe[i]];
       end;
```

SourceCode 4.12 operasi pertama pada mesin dan *job*

Pada SourceCode 4.12 jika jadwal yang sedang dihitung makspannya menangalami penambahan *job* baru (`status =2`) maka *start* mesinnya (`StartMesin[NoMesin[i],MesinKe[i]]`) adalah waktu penambahan, jika tidak maka waktu *start*nya adalah 0. Waktu *finish* adalah lama waktu pengerjaan operasi (`MCost[OperasiKe[i],Job[i]]`). Waktu *start* dan *finish job* (`StartJob[Job[i],OperasiKe[i]]`, dan `FinishJob[Job[i],OperasiKe[i]]`)digunakan sebagai alat bantu menentukan penghitungan operasi berikutnya.

Pada SourceCode 4.13 digunakan untuk menyelesaikan kondisi ke dua dimana mesin pertama kali digunakan tetapi yang operasi yang dikerjakan bukanlah operasi pertama pada *job* tersebut. Untuk jadwal dengan penambahab *job* baru (`status=2`) jika operasi terakhir pada *job* itu memiliki waktu *finish* lebih kecil dari waktu penambahan (`OperasiKe[i]-1]<WaktuTambahJob`), maka *start* mesinnya adalah waktu penambahan *job* baru, jika tidak maka waktu *start* mesin adalah waktu *finish* mesin operasi terakhir yang dianggap telah selesai dikerjakan.

```
// mesin digunakan pertama, tapi bukan operasi pertama
untuk job itu
    if  (MesinKe[i]=1)and (OperasiKe[i]<>1) then
      begin
        if (Status=2)and(FinishJob[Job[i],
                  OperasiKe[i]-1]<WaktuTambahJob then
          begin
            StartMesin [NoMesin[i],MesinKe[i]]:=
                                        WaktuTambahJob;
          end
        else
           begin
              StartMesin[NoMesin[i],MesinKe[i]]:=
                     FinishJob[Job[i],OperasiKe[i]-1];
           end;
           FinishMesin[NoMesin[i],MesinKe[i]]:=
                     StartMesin[NoMesin[i],MesinKe[i]]+
                            MCost[OperasiKe[i],Job[i]];
           StartJob[Job[i],OperasiKe[i]]:=
                     StartMesin[NoMesin[i],MesinKe[i]];
           FinishJob[Job[i],OperasiKe[i]]:=
                    FinishMesin[NoMesin[i],MesinKe[i]];
      end ;
```

SourceCode 4.13 operasi pertama pada mesin tapi tidak pada *job*

```
//Operasi pertama tapi mesin sudah digunakan sebelumnya
    if  (MesinKe[i]<>1)and (OperasiKe[i]=1) then
      begin
         if (Status=2) and (FinishMesin[NoMesin[i],
                MesinKe[i]-1]< WaktuTambahJob)  then
            begin
              StartMesin[NoMesin[i],MesinKe[i]]:=
                                        WaktuTambahJob;
            end
        else
          begin
               StartMesin[NoMesin[i],MesinKe[i]]:=
                   FinishMesin[NoMesin[i],MesinKe[i]-1];
          end;
          FinishMesin[NoMesin[i],MesinKe[i]]:=
                      StartMesin[NoMesin[i],MesinKe[i]]+
                             MCost[OperasiKe[i],Job[i]];
          StartJob[Job[i],OperasiKe[i]]:=
                      StartMesin[NoMesin[i],MesinKe[i]];
          FinishJob[Job[i],OperasiKe[i]]:=
                     FinishMesin[NoMesin[i],MesinKe[i]];
      end
```

SourceCode 4.14 operasi pertama pada *job* tapi tidak pada mesin

SourceCode 4.14 menyelesaian kondisi ke-3 dimana operasi yang dikerjakan oleh mesin adalah operasi pertama pada *job* itu. Sedangkan mesin sudah melakukan pemrosesan operasi dari *job* yang lain sebelumya maka, waktu star mesin adalah waktu *finish* mesin untuk operasi yang sebelumnya. Untuk jadwal dengan penambahan *job* baru (`status=2`) maka dilakukan pengecekan lagi apakah waktu *finish* mesin untuk operasi terakhir (sebelum ada penambahan *job* baru) lebih kecil dari waktu penambahan (*Finish*`Mesin[NoMesin[i],MesinKe[i]-1]< WaktuTambah`*Job*) jika benar maka waktu *start* untuk operasi tersebut adalah waktu datang *job* baru (`WaktuTambah`*Job*).

```
//bukan operasi pertama dan mesin sudah digunakan
sebelumnya
    else
      begin
        // cari Start mesin
        if (FinishJob[Job[i],OperasiKe[i]-1])>
             (FinishMesin[NoMesin[i],MesinKe[i]-1]) then
           begin
              StartMesin[NoMesin[i],MesinKe[i]]:=
                        FinishJob[Job[i],OperasiKe[i]-1];
           end
        else
          begin
             StartMesin[NoMesin[i],MesinKe[i]]:=
                   FinishMesin[NoMesin[i],MesinKe[i]-1];
          end;
        //cek StartMesin
        if (Status=2) and (StartMesin[NoMesin[i],
                        MesinKe[i]]< WaktuTambahJob)then
          begin
              StartMesin[NoMesin[i],MesinKe[i]]:=
                                          WaktuTambahJob;
          end;
          FinishMesin[NoMesin[i],MesinKe[i]]:=
                      StartMesin[NoMesin[i],MesinKe[i]]+
                             MCost[OperasiKe[i],Job[i]];
          StartJob[Job[i],OperasiKe[i]]:=
                      StartMesin[NoMesin[i],MesinKe[i]];
          FinishJob[Job[i],OperasiKe[i]]:=
                     FinishMesin[NoMesin[i],MesinKe[i]];
     end;
```

SourceCode 4.15 sama-sama bukan operasi pertama pada mesin dan *job*

Pada kondisi ke-4 dimana mesin sudah digunakan sebelumnya dan operasi pada *job* yang sedang dikerjakan adalah bukan operasi pertama. Maka waktu *start* mesin didapat dengan membandingkan waktu *finish* mesin untuk operasi sebelumnya dengan waktu *finish job* operasi sebelumnya (*FinishJob*[*Job*[i], OperasiKe[i]-1])>(*Finish*Mesin[NoMesin[i],MesinKe[i]-1]) yang terbesar digunakan sebagai waktu *start* mesin. Jika jadwal yang sedang diproses mengalami penambahan *job* baru (Status=2) maka, dilakukan perbandingan kembali antar waktu *start* yang ada dengan waktu datang *job* baru (*Start*Mesin[NoMesin[i],MesinKe[i]] < WaktuTambah*Job*) yang memiliki nilai terbesar dipilih sebagai waktu *start* mesin. Waktu *finish* mesin adalah waktu *start* mesin ditambah waktu proses operasi (*Start*Mesin[NoMesin[i],MesinKe[i]] + MCost[OperasiKe[i],*Job*[i]]).

4.2.6 Pencarian nilai *makespan*

Setelah semua operasi dihitung waktu *start* dan *finish*nya maka *makespan* didapatkan dengan mencari waktu *finish* yang terbesar dari tiap-tiap mesin seperti pada SourceCode 4.16. proses pencariannya mula–mula *finish* mesin pertama untuk operasi terakhir (*Finish*Mesin[1,n*Job*]) dianggap sebagai *makespan* (cost). Kemudian dilakukan pembandingan dengan semua mesin yang ada, jika mesin yang berikutnya nilai *finish* mesin untuk operasi terakhir lebih besar daripada *makespan* (cost) maka nilainya diupdate.

```
cost:=FinishMesin[1,nJob];  //mencari nilai cost max
  for i :=2  to nMesin  do
    begin
      if FinishMesin[i,nJob]> cost then
          cost:=FinishMesin[i,nJob]
      else
          cost:=cost;
    end;
```

SourceCode 4.16. mencari *makespan*

4.2.7 Penghitungan Fitnes

Nilai *fitness* bergantung pada tujuan dari proses optimasi (maksimasi atau minimasi). Pada kasus *job shop* optimasi yang dimaksud adalah minimasi sehingga penghitungan nilai *fitness* adalah 1/ *makespan* (Cost) seperti pada SourceCode 4.17. Nilai

makespan didapatkan dengan memanggil fungsi penghitungan *makespan* yaitu `HitungFitnes(ind)`.

```
begin
    Ind.Cost := HitungFitnes(ind);
    Ind.Fitness := 1/(0.0001+Ind.cost)
end;
```

SourceCode 4.17 penghitungan nilai *fitness*

4.2.8 Cek data jadwal Awal

Pada kondisi dimana terjadi penambahan *job* baru maka untuk pendekatan dengan meneruskan operasi yang sudah selesai dikerjakan dan menjadwal ulang *job* baru dan operasi yang belum selesai dikerjakan perlu dilakukan pengecekan operasi-operasi yang sudah selesai dikerjakan. Pengecekan dilakukan pada waktu *start job* dan *start* mesin dengan kriteria waktu *start* pada operasi tersebut lebih kecil dari waktu datang *job* baru (`WaktuPenambahanJob`). Implementasi program pengecekan jadwal untuk waktu *start job* ada pada SourceCode 4.18.

```
// cek data terdahulu (jadwal awal yang telah terbentuk)
  WaktuPenambahanJob:= (WaktuPenambahan/100)*CostAwal;
  //  % waktu penambahan
  for a:=1 to nJobPertama do
    begin
    temp:=0;
       for b:=1 to nMesin do
        begin
          if StartJob[a,b]<WaktuPenambahanJob then
           begin
            temp:=temp+1;
           end;
        end;
        JOSelesai[a]:= temp;
    end;
```

SourceCode 4.18 proses cek *start job*

Kegunaan dari pengecekan waktu *start job* adalah untuk menentukan berapa operasi yang telah selesai dikerjakan pada tiap-tiap *job*. Hasil pengecekan waktu *start job* (`JOSelesai[a]`) digunakan untuk menentukan berapa jumlah operasi yang akan dijadwalkan kembali (`JOperasi[a]`). Pada SourceCode 4.19 melakukan pengecekan waktu *start* mesin, hal ini dilakukan untuk mengetahui

tiap tiap mesin (MOSelesai[a]) sudah digunakan berapa kali. Hal ini diperlukan untuk menentukan urutan pengerjaan operasi pada jadwal yang baru. Kriteria pengecekannya sama yaitu *start* mesin yang memiliki nilai dibawah waktu penambahan *job* baru (*Start*Mesin[a,b] < WaktuPenambahan*Job*).

```
for a:=1 to nMesin do
 begin
  temp:=0;
  for b:=1 to nJobPertama do
   begin
    if StartMesin[a,b]<WaktuPenambahanJob then
     begin
      temp:=temp+1;
     end;
   end;
   MOSelesai[a]:=temp;
end;
```

SourceCode 4.19 proses cek *start* mesin

4.3 Implementasi Antarmuka

Berdasarkan rancangan antarmuka pada subbab 3.2.10 maka dihasilkan antarmuka pada gambar 4.1.

4.4 Implementasi Uji Coba

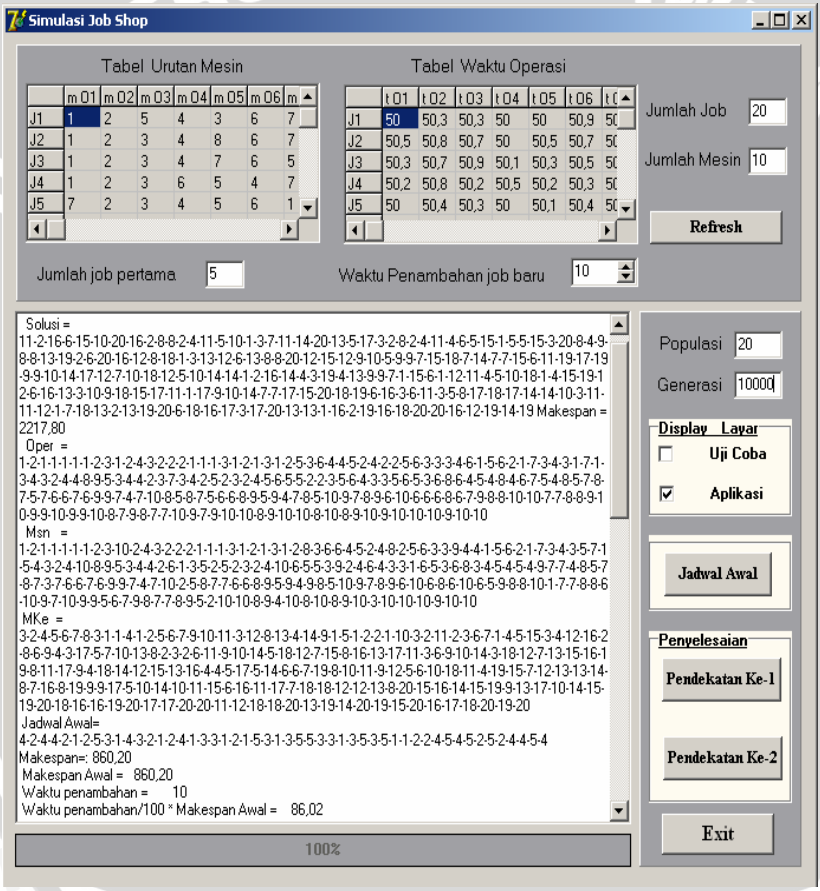
4.4.1 Skenario Uji Coba

Skenario yang digunakan sama dengan pembahasan pada subbab 3.3.2, dimana akan diterapkan beberapa aturan diantaranya banyaknya *job* bervariasi (16,20 dan 32) pada tiap tahapnya. Banyaknya mesin bervariasi (6, 10 dan 14) pada tiap tahapnya, waktu penambahan *job* baru bervariasi (10%, 20%, 30%, 40%, 50%, 60% dan 70% dari *makespan* jadwal awal). Jumlah *job* pertama ada dua kondisi yaitu 1/4 dan 3/4 dari total *job*. Tiap percobaan dilakukan 5 kali pengulangan sehingga nantinya akan terdapat 125 kali percobaan pada tiap tahapnya. Sedangkan untuk waktu proses tiap *job* akan dibangkitkan nilai random antara 50 sampai 50.9. Parameter algoritma genetik ditentukan sebagai berikut:

- Jumlah generasi = 10000
- Jumlah populasi = 20

- Peluang mutasi = 0.05
- Peluang *crossover* = 0.95

Pada hasil uji coba, penyelesain masalah *job shop* dinamis dengan mengabaikan operasi-operasi yang dianggap telah selesai dikerjakan dan menjadwal ulang pada semua *job* disebut dengan pendekatan pertama. Sedangkan cara penyelesaian masalah *job shop* dinamis dengan tetap mempertahankan jadwal awal (operasi yang dianggap selesai) disebut dengan pendekatan kedua.



Gambar 4.1 Implementasi antarmuka

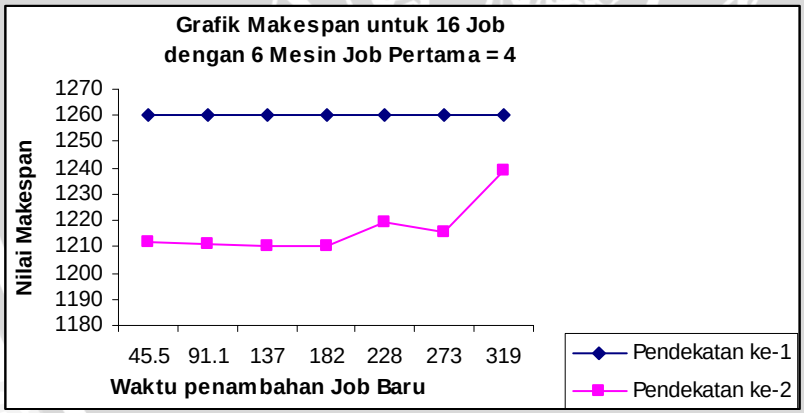
4.4.2 Hasil Uji Coba

Dari skenario uji coba sebelumnya dimana masing-masing kombinasi dilakukan 5 kali percobaan dan diambil nilai terbaik, maka didapatkan hasil uji coba sebagai berikut:

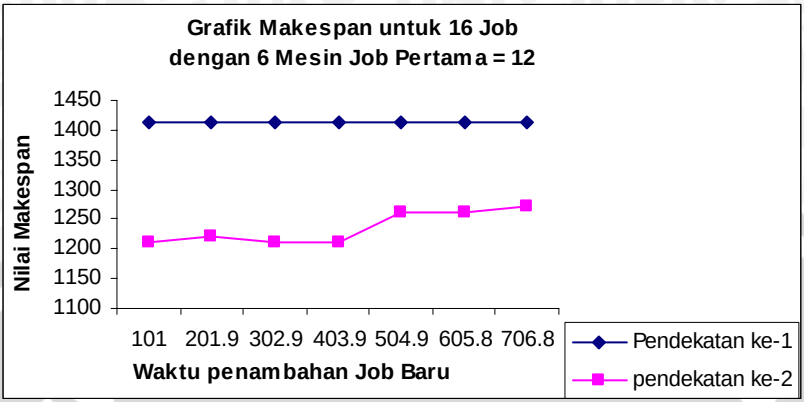
4.4.2.1 Jumlah *job* =16 dan jumlah mesin = 6

Hasil uji coba dengan menggunakan *job* pertama sebesar ¼ dari total *job* (*job* pertama = 4) digambarkan pada Gambar 4.2, sedangkan hasil uji coba dengan menggunakan *job* pertama sebesar ¾ dari total *job* (*job* pertama = 12) digambarkan pada Gambar 4.3. Nilai *makespan* pada tiap-tiap percobaan ada di lampiran. Untuk *job* pertama sebesar 4 ada pada Lampiran 1 dan *job* pertama sebesar 12 ada pada Lampiran 2.

Dari Gambar 4.2 dan Gambar 4.3 dapat diketahui bahwa nilai *makespan* pendekatan pertama tak mengalami perubahan dengan kenaikan nilai waktu penambahan *job* baru. Nilai *makespan* pendekatan kedua mengalami kenaikan mengikuti kenaikan nilai waktu penambahan *job* baru.



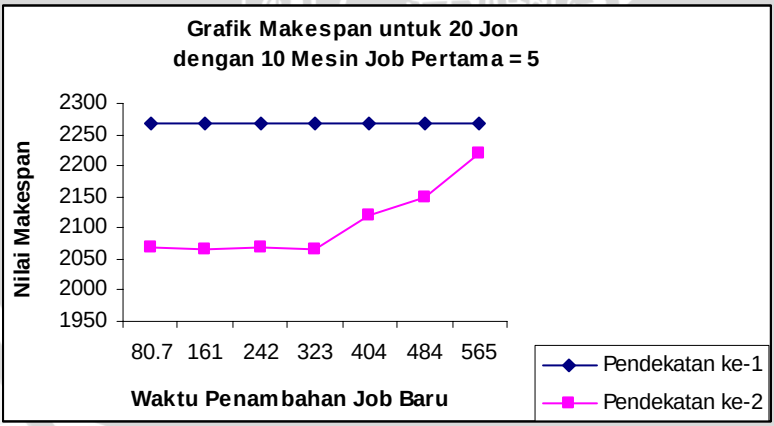
Gambar 4.2 Grafik untuk 16 *job* dengan 6 mesin *job* pertama = 4



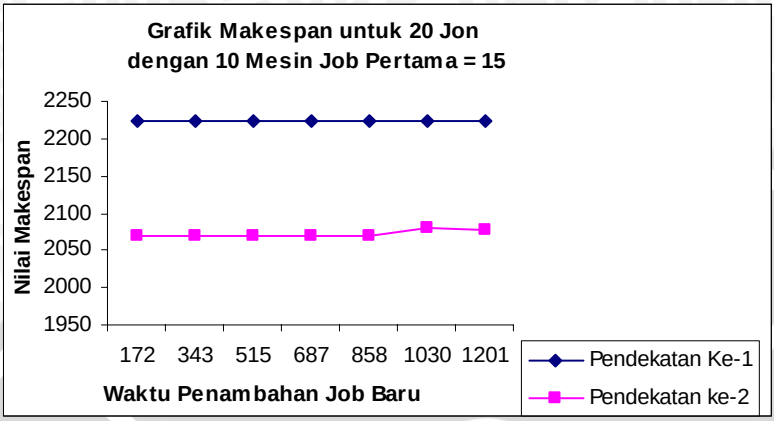
Gambar 4.3 Grafik untuk 16 *job* dengan 6 mesin *job* pertama = 12

4.4.2.2 Jumlah *job* =20 dan jumlah mesin = 10

Hasil uji coba dengan menggunakan *job* pertama sebesar ¼ dari total *job* (*job* pertama = 8) digambarkan pada Gambar 4.4, sedangkan hasil uji coba dengan menggunakan *job* pertama sebesar ¾ dari total *job* (*job* pertama = 16) digambarkan pada Gambar 4.5. Nilai *makespan* pada tiap-tiap percobaan ada di lampiran. Untuk *job* pertama sebesar 8 ada pada Lampiran 3 dan *job* pertama sebesar 16 ada pada Lampiran 4.



Gambar 4.4 Grafik untuk 20 *job* dengan10 mesin *job* pertama = 5

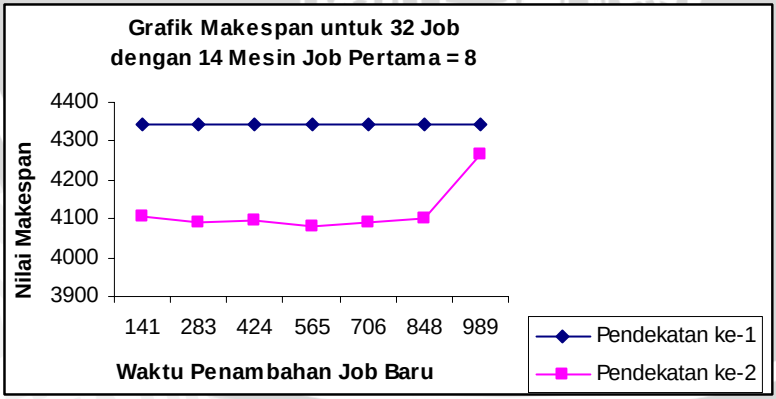


Gambar 4.5 Grafik untuk 20 *job* dengan 10 mesin *job* pertama = 15

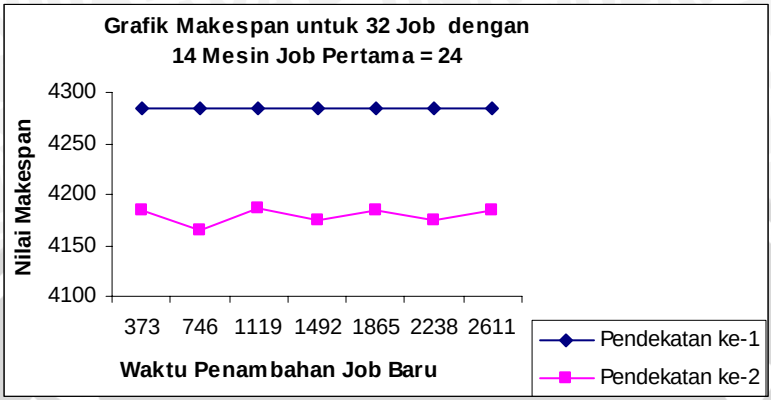
Dari Gambar 4.4 dan Gambar 4.5 dapat diketahui bahwa nilai *makespan* pendekatan kedua meningkat searah dengan kenaikan nilai waktu penambahan *job* baru. Sedangkan pendekatan pertama relatif tetap.

4.4.2.3 Jumlah *job* = 32 dan jumlah mesin = 14

Hasil uji coba dengan *job* pertama sebesar 8 digambarkan pada Gambar 4.4, sedangkan hasil uji coba dengan *job* pertama sebesar 16 digambarkan pada Gambar 4.5. Nilai *makespan* pada tiap-tiap percobaan ada di lampiran. Untuk *job* pertama sebesar 8 ada pada Lampiran 5 dan *job* pertama sebesar 16 ada pada Lampiran 6.



Gambar 4.6 Grafik untuk 32 *job* dengan 14 mesin *job* pertama = 8



Gambar 4.7 Grafik untuk 32 *job* dengan 14 mesin *job* pertama = 24

Nilai *makespan* pendekatan pertama tetap tak mengalami perubahan dengan kenaikan nilai waktu penambahan *job* baru, hal tersebut terlihat pada Gambar 4.6. Pada Gambar 4.7. Sedangkan pendekatan kedua relatif stabil hanya ada beberapa titik yang mengalami kenaikan dengan adanya peningkatan waktu penambahan *job* baru.

4.4.3 Analisis Hasil Uji Coba

Dari hasil uji coba pada subbab 4.4.2, dapat diketahui bahwa pendekatan pertama memiliki nilai *makespan* yang tak terpengaruh dengan kenaikan nilai waktu penambahan *job* baru. Pendekatan kedua memiliki hasil uji coba yang sedikit berbeda pada setiap kombinasi yang diterapkan. Untuk kombinasi 16 *job* dengan 6 mesin, pada penggunaan *job* pertama sebesar 4 (¼ dari total *job*) nilai *makespan* pada beberapa titik penambahan *job* baru mengalami penurunan tetapi dilain titik juga terjadi kenaikan. Pada penggunaan *job* pertama sebesar 12 (¾ dari total *job*) kenaikan nilai *makespan* terjadi pada semua titik penambahan *job* baru. Untuk kombinasi 20 *job* dan 10 mesin, pada penggunaan *job* pertama sebanyak 5 (¼ dari total *job*), nilai *makespan* cenderung naik bahkan pada waktu penambahan diatas 50% mengalami kenaikan yang cukup besar. Berbeda yang terjadi pada penggunaan *job* awal sebesar 15 (¾ dari total *job*) nilai *makespan* relatif stabil dengan kenaikan yang tak terlalu besar. Begitu pula yang terjadi pada penerapan kombinasi 32

job dengan 14 mesin untuk penggunaan *job* pertama sebesar 8 (¼ dari total *job*) nilai *makespan* mengalami peningkatan sedangkan pada penggunaan *job* pertama sebesar 24 (¾ dari total *job*) nilai *makespan* relatif stabil.

Tingkat kenaikan nilai *makespan* pendekatan kedua pada penggunaaan jumlah *job* pertama sebesar ¾ dari total *job* relatif lebih kecil jika dibandingkan dengan penggunaan *job* pertama sebesar ¼ dari total *job* .

Pada pendekatan pertama nilai *makespan* relatif tetap pada setiap titik waktu penambahan *job* baru terjadi karena, yang menjadi penentua nilai *makespan* adalah jadwal awal yang dibentuk sebelum terjadi penambahan *job* baru. Sesuai dengan rancangan pada Bab III bahwa pada pendekatan pertama ini hanya menjadwalkan operasi dari job baru jadwal lama diabaikan dan tak diikutkan dalam penjadwalan yang baru. Karena waktu penambahan job baru bergantung dari jadwal awal maka hal ini tak berpengaruh pada pendekatan pertama.

Pada pendekatan kedua adanya kenaikan nilai *makespan* pada sebagian, atau bahkan pada semua titik waktu penambahan *job* baru terjadi karena pendekatan kedua ini hanya menjadwalkan operasi yang belum selesai dikerjakan. Sehingga besar kemungkinan jadwal awal yang terbentuk tidaklah memiliki susunan pengerjaan tiap operasi yang baik, sehingga dengan makin bertambahnya waktu penambahan *job* baru maka semakin banyak waktu terbuang yang mengakibatkan kenaikan nilai *makespan.*

Perbedaan tingkat penurunan nilai *makespan* pada penggunaan *job* awal sebesar ¼ dan ¾ dari total *job* terjadi karena, adanya perbedaan perbandingan antara jumlah mesin yang tersedia dengan jumlah *job* yang akan dijadwalkan. Seperti di ketahui setiap *job* pada sistem produksi *job shop* memiliki urutan operasi yang berbeda , tiap operasi hanya bisa dikerjakan jika operasi sebelumnya telah selesai dikerjakan. Sebagai contoh digunakan kombinasi 20 *job* dengan 8 mesin, pada *job* pertama sebesar 5 kalaupun operasi pertama pada tiap *job* berbeda misalkan:

job 1 operasi 1 di mesin 1,
job 2 operasi 1 di mesin 2,
job 3 operasi 1 di mesin 3,
job 4 operasi 1 di mesin 4,

job 5 operasi 1 di mesin 5, maka mesin 6, 7 dan 8 akan menganggur. Ketiga mesin yang menganggur tadi (mesin 6, 7 dan 8) akan menunggu operasi pertama selesai dikerjakan. Hal ini tentunya akan membuat waktu penyelesian semua operasi bertambah. Hal yang berbeda terjadi ketika *job* pertama sebesar 15, dengan 8 mesin yang tersedia besar kemungkinan semua mesin akan bekerja. Semakin berkurangnya mesin yang menganggur pada pengerjaan jadwal awal berarti semakin banyak pula operasi yang selesai pada tiap titik penambahan *job* baru, hal inilah yang membuat nilai *makespan* pada penggunaan *job* pertama sebesar ¾ selalu lebih besar dibandingkan dengan penggunaan *job* pertama sebesar ¼..

Peningkatan kompleksitas *job shop* tidak terlalu berpengaruh kepada hasil uji coba (*makespan* yang didapatkan). Pengaruh yang dimaksud adalah perubahan pergerakan data. Tiap-tiap pendekatan memiliki kecenderungan pergerakan data yang relatif sama meskipun jumlah operasi (kompleksitas *job shop*) meningkat. Hal ini dapat dilihat dengan jalan membandingkan nilai *makespan* untuk kondisi yang sama dengan kompleksitas *job shop* yang berbeda. Sebagai contoh digunakan data pada Gambar 4.4 dan Gambar 4.5 (kombinasi 20 *job* dengan 8 mesin) dan data pada Gambar 4.6 dan Gambar 4.7 (kombinasi 32 *job* dengan 14 mesin). Dari kedua tabel tersebut diketahui bahwa baik pendekatan pertama mupun pendekatan kedua memiliki pergerakan data yang relatif sama, pendekatan pertama relatif tetap dan pendekatan kedua mengalami kenaikan nilai *makespan.*

BAB V
PENUTUP

5.1 Kesimpulan

Dari hasil uji coba yang telah dilakukan, maka dapat diambil kesimpulan antara lain :

1. Model algoritma genetik (kromosom, *crossover* dan mutasi yang dirancang), berhasil menyelesaikan permasalahan *job shop* dinamis.
2. Peningkatan waktu penambahan *job* baru tak berpengaruh terhadap nilai *mekespan* (waktu yang dibutuhkan untuk menyelesaikan semua operasi) pendekatan pertama yaitu pendekatan dengan mengabaikan operasi yang telah selesai dikerjakan.
3. Pada pendekatan kedua (pendekatan dengan mempertahankan jadwal awal dan melakukan penjadwalan ulang terhadap operasi dari *job* baru dan sisa operasi dari jadwal awal yang belum diselesaikan) peningkatan waktu penambahan *job* baru mengakibatkan peningkatan nilai *makespan*.
4. *Makespan* pendekatan pertama dan kedua tak terlalu terpengaruh dengan peningkatan kompleksitas *job shop*.

5.2 Saran

Saran yang dapat diberikan setelah pengerjaan skripsi ini adalah :

1. Dari hasil uji coba terdapat beberapa pergerakan data hasil uji coba yang tak seragam (tingkat kenaikan dan penurunan nilai *makespan*), hal ini mungkin bisa diperbaiki dengan pemilihan operator *crossover* dan mutasi yang berbeda.
2. Penelitian ini dapat dikembangkan lagi dengan menggunakan algoritma heuristik yang lain seperti *simulated annealing* dan *tabu search.*

DAFTAR PUSTAKA

Bierwirth Christian, Dirk C. Mattfeld .1999. *Production Scheduling and Rescheduling with Genetic Algorithms,* University of BremenDepartment of Economics. Massachusetts Institute of Technology Evolutionary Computation 7(1): 1-17.

Budhi Gregorius S.2003 . *Pendekatan Activity-Based Costing Dan MetodePencarian Heuristic Untuk Menyelesaikan Problem Pemilihan Peralatan Pada Flexible Manufacturing Systems (Fms)*Petra, Surabaya.

Fang, Hsiao-Lan.1994. *Genetic Algorithms in Timetabling and Scheduling,* Ph.D. Dissertation, Department of Artificial Intelligence, University of Edinburgh.

Gamma. 2006. *Algoritma Penjadwalan Produksi Pada Lingkungan Mesin Job ShopDengan Minimalisasi Rataan Waktu Tunggu Operasi,* Bandung.

Hetti Mauluti N, Muflihah Agustin, Nilah Qurrotu A, Aulia Musyayadah. 2006, *Penerapan Metode Campbell Dudeks And Smith (Cds Methods) Dalam Penentuan Urutan Pengerjaan Produk Untuk Meminimalkan Waktu Penyelesaian Produk (Studi Kasus Pada Pt. Kemajuan Industrindo),* Jurusan Teknik Industri, Universitas Muhammadiyah Malang, Malang .

Keith Schmidt. 2001. *Using Tabu Search to Solve the Job Shop Scheduling Problem with Sequence Dependent Setup Times*.

Marko Snoek. 1999. *Anticipation Optimization in Dynamic Job Shops* , Department of Computer Science University of Twente, Enschede, The Netherlands .

Megan Thurber. 2004. *Contrasting Search Algorithms implemented on the Job-Shop Scheduling Problem,* Department of Computer Science Colorado State University .

Muhammet Yaman , Hamit Saruhan , Faruk Mendi. 2004, *Power Loss Optimization Of A Worm Gear Mechanism By Using Genetic Algorithm*, Technical Education Faculty, Abant Izzet Baysal University, Duzce, Turkey, M.S., Dr.Technical Education Faculty, Gazi University, Ankara, Turkey, Dr.

Noversada M.2006. *Algoritma Tabu search dan penggunaannya dalam penyelesaian job shop scheduling*, Laboraturium dan ilmu rekayasa ITB, Bandung .

Retno Sari Dewi Dian.2005. *Pengembangan Algoritma Penjadualan Produksi Job Shop Untuk Meminimumkan Total Biaya Earliness Dan Tardiness*, jurnal teknik industri Widya Mandala, Surabaya.

Saputro Nico, Yento.2003. *Penjadwalan Job Shop Dinamis Non Deterministik*, Universitas Katolik Parahyangan .

Suyanto. 2005. *Algoritma Genetik dalam Matlab*, Andi Offset, Yogyakarta .

LAMPIRAN

Lampiran 1 Data uji coba untuk total *job* = 16 dengan 6 mesin dan jumlah *job* pertama = 4

Waktu penambahan job baru	Pendekatan ke-1	Pendekatan ke-2
100.97	1260	1211.9
201.94	1260	1211.3
302.91	1260	1210.3
403.88	1260	1210.5
504.85	1260	1219.2
605.82	1260	1215.3
706.79	1260	1239.1

Lampiran 2 Data uji coba untuk total *job* = 16 dengan 6 mesin dan jumlah *job* pertama =12

Waktu penambahan job baru	Pendekatan ke-1	Pendekatan ke-2
100.97	1413.4	1210.4
201.94	1413.4	1220.6
302.91	1413.4	1210.3
403.88	1413.4	1210.8
504.85	1413.4	1260.5
605.82	1413.4	1260.5
706.79	1413.4	1270.6

Lampiran 3 Data uji coba untuk total *job* = 20 dengan 10 mesin dan jumlah *job* pertama = 5

Waktu penambahan job baru	Pendekatan ke-1	Pendekatan ke-2
80.7	2268.8	2067.9
161.4	2268.8	2066.3
242.1	2268.8	2067.4
322.8	2268.8	2066.6
403.5	2268.8	2118.6
484.2	2268.8	2150
564.9	2268.8	2218.8

Lampiran 4 Data uji coba untuk total *job* = 20 dengan 10 mesin dan jumlah *job* pertama =15

Waktu penambahan job baru	Pendekatan ke-1	Pendekatan ke-2
171.64	2223	2069.1
343.28	2223	2069.1
514.92	2223	2069.3
686.56	2223	2069
858.2	2223	2069.3
1029.84	2223	2079.44
1201.48	2223	2078

Lampiran 5 Data uji coba untuk total *job* = 32 dengan 14 mesin dan jumlah *job* pertama = 8

Waktu penambahan job baru	Pendekatan ke-1	Pendekatan ke-2
141.27	4342.4	4107.6
282.54	4342.4	4090.3
423.81	4342.4	4097.41
565.08	4342.4	4079.68
706.35	4342.4	4091.55
847.62	4342.4	4099.02
988.89	4342.4	4267.59

Lampiran 6 Data uji coba untuk total *job* = 32 dengan 14 mesin dan jumlah *job* pertama = 24

Waktu penambahan job baru	Pendekatan ke-1	Pendekatan ke-2
372.99	4284.3	4184.7
745.98	4284.3	4165.6
1118.97	4284.3	4186.3
1491.96	4284.3	4175.2
1864.95	4284.3	4184.85
2237.94	4284.3	4173.64
2610.93	4284.3	4183.93

KATA PENGANTAR

Puji syukur penulis panjatkan kehadirat Allah SWT, karena atas segala rahmat dan limpahan hidayahnya, Skripsi yang berjudul "*Pengaruh Waktu Penambahan Job Baru Pada Penjadwalan Job Shop Dinamis Menggunakan Algoritma Genetik*" ini dapat diselesaikan. Skripsi ini disusun dan diajukan sebagai syarat untuk memperoleh gelar sarjana pada program studi Ilmu Komputer, Jurusan Matematika, Fakultas MIPA, Universitas Brawijaya.

Dalam penyelesaian skripsi ini, penulis telah mendapat begitu banyak bantuan dari banyak pihak. Atas bantuan yang telah diberikan, penulis ingin menyampaikan penghargaan dan ucapan terima kasih kepada:

1. Drs. Ahmad Ridok, M.Kom selaku pembimbing pertama. Terima kasih atas semua saran, bantuan, kritikan, waktu, dorongan semangat dan bimbingannya.
2. Bayu Rahayudi, ST.,MT selaku pembimbing kedua. Terima kasih atas kesediaannya membimbing dalam penulisan skripsi ini sehingga mudah dan enak dibaca.
3. Wayan F. Mahmudy, S.Si., MT selaku Ketua Program Studi Ilmu Komputer Universitas Brawijaya Malang.
4. Dian Eka R. S.Si.,M.Kom selaku penasehat akademik yang banyak memberikan masukan demi kelancaran perkuliahan.
5. Bapak, ibu dan saudaraku yang senantiasa berdoa dan memberi dukungan dan semangat.
6. Semua teman-teman Ilmu Komputer angkatan 2003 dan 2004 terima kasih atas dukungan dan doanya.
7. Semua pihak lain yang telah membantu hingga terselesaikannya penyusunan penulisan tugas akhir ini.

Semoga penulisan skripsi ini bermanfaat bagi pembaca sekalian. Penulis menyadari bahwa skripsi ini masih jauh dari kesempurnaan sehingga penulis mengharapkan masukan dan saran yang membangun dari pembaca.

Malang, 7 Mei 2008

Penulis

DAFTAR ISI

Daftar Gambar

Daftar Tabel

Daftar Source Code

Daftar lampiran