PERBANDINGAN KINERJA ALGORITMA GENETIKA DAN ALGORITMA SIMULATED ANNEALING PADA FLOW SHOP SCHEDULING UNTUK PERHITUNGAN MAKESPAN DAN TOTAL FLOWTIME

SKRIPSI

oleh:

TEGUH IMANTOKO

0310963037-96

PROGRAM STUDI ILMU KOMPUTER

JURUSAN MATEMATIKA

FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM

UNIVERSITAS BRAWIJAYA

2007

PERBANDINGAN KINERJA ALGORITMA GENETIKA DAN ALGORITMA SIMULATED ANNEALING PADA FLOW SHOP SCHEDULING UNTUK PERHITUNGAN MAKESPAN DAN TOTAL FLOWTIME

SKRIPSI

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

oleh:

TEGUH IMANTOKO
0310963037-96

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

2007

LEMBAR PENGESAHAN SKRIPSI

PERBANDINGAN KINERJA ALGORITMA GENETIKA DAN ALGORITMA SIMULATED ANNEALING PADA FLOW SHOP SCHEDULING UNTUK PERHITUNGAN MAKESPAN DAN TOTAL FLOWTIME

Oleh:
TEGUH IMANTOKO
0310963037-96

Setelah dipertahankan di depan Majelis Penguji
Pada tanggal 8 November 2007
dan dinyatakan memenuhi syarat untuk memperoleh
gelar Sarjana dalam bidang Ilmu Komputer

Pembimbing I	Pembimbing II
Wayan Firdaus M., SSi., MT	Dian Eka R., SSi., M.Kom
NIP. 132 158 724	NIP. 132 300 224

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 : Teguh Imantoko
NIM : 0310963037-96
Jurusan : Matematika
Penulis Tugas Akhir berjudul : Perbandingan Kinerja Algoritma Genetika dan Algoritma Simulated Annealing pada Flow Shop Scheduling untuk Perhitungan Makespan dan Total Flowtime

Dengan ini menyatakan bahwa :

1. **Isi dari Tugas Akhir 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 Tugas Akhir ini.**
2. **Apabila dikemudian hari ternyata Tugas Akhir 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, 8 November 2007
Yang menyatakan,

(Teguh Imantoko)
NIM. 0310963037

PERBANDINGAN KINERJA ALGORITMA GENETIKA DAN ALGORITMA SIMULATED ANNEALING PADA FLOW SHOP SCHEDULING UNTUK PERHITUNGAN MAKESPAN DAN TOTAL FLOWTIME

ABSTRAK

Penelitian difokuskan pada perbandingan algoritma genetika dan algoritma simulated annealing untuk menghitung prosentase keunggulan dan waktu proses antar algoritma tersebut. Tujuan penelitian adalah untuk mengetahui algoritma mana yang lebih baik diterapkan pada permasalahan *flow shop scheduling* yang disimulasikan untuk perhitungan nilai *makespan* dan total *flowtime*.

Proses simulasi dari program penjadwalan *flow shop* tersebut dilakukan dengan bermacam-macam kombinasi jumlah job dan mesin yang berbeda-beda. Dari hasil simulasi, score algoritma genetika (GA) lebih unggul dari *simulated annealing* (SA). Rata-rata prosentase keunggulan SA masih kalah dibandingkan GA. Prosentase keunggulan dari SA hanya unggul sedikit dibanding GA pada kombinasi jumlah job yang tidak terlalu besar. Keunggulan SA hanya pada nilai *makespan* saja. Sebaliknya, rata-rata prosentase keunggulan dari GA terpaut cukup jauh dibanding SA. Keunggulan GA ini dikarenakan nilai total *flowtime* yang jauh lebih baik dari SA. Lain halnya dengan waktu proses yang dihasilkan, dimana waktu yang dibutuhkan algoritma SA lebih baik (lebih cepat) dari waktu yang dibutuhkan GA.

Kata Kunci : Algoritma Genetika, Algoritma Simulated Annealing, Flow Shop Scheduling, Makespan, Total Flowtime

GENETIC ALGORITHM AND SIMULATED ANNEALING ALGORITHM WORKS COMPARISON IN FLOW SHOP SCHEDULING FOR MAKESPAN AND TOTAL FLOWTIME CALCULATION

ABSTRACT

The research is focused on comparing genetic algorithm and *simulated annealing* algorithm to calculate the percentage of advantage and the process time between each algorithm. The aim of the research is to find out which algorithm works best when implemented on *flow shop scheduling* problem simulated for *makespan* and total *flowtime* calculation.

The simulation process of the *flow shop scheduling* program is done with various combination of job count and different type of machines. From the simulation results, Genetic Algorithm (GA) is found to be superior to the *Simulated Annealing* (SA) algorithm. The mean percentage of *Simulated Annealing*'s advantage is lower than GA's. SA's percentage advantage remains small compared to GA and it only happens on a small combination of job count. SA's only advantage is on the *makespan* score. In the other hand, the mean percentage of GA is far above SA's. The GA's advantage is due to better total *flowtime* score compared to SA. This is different in regards of the processing time result, where the time needed by SA algorithm is better (it works faster) than what is needed by GA.

Keyword : Genetic Algorithm, Simulated Annealing Algorithm,
Flow Shop Scheduling, Makespan, Total Flowtime

Kata Pengantar

Alhamdulillahi rabbil ‘alamin. Puji syukur penulis panjatkan kehadirat Allah SWT, karena atas segala rahmat dan limpahan hidayah-Nya, Tugas Akhir yang berjudul “*Perbandingan Kinerja Algoritma Genetika dan Algoritma Simulated Annealing pada Flow Shop Scheduling untuk Perhitungan Makespan dan Total Flowtime*” ini dapat berjalan dengan baik. Tugas Akhir ini disusun dan diajukan sebagai syarat untuk memperoleh gelar sarjana pada program studi Ilmu Komputer, jurusan Matematika, fakultas MIPA, universitas Brawijaya.

Salawat serta salam semoga tetap tercurahkan kepada Baginda Rasulullah Muhammad SAW, makhluk paling mulia yang senantiasa memberikan cahaya petunjuk, seorang uswatun hasanah yang telah membawa agama Allah yaitu agama Islam menjadi agama yang Rahmatan Lil ‘Alamin.

Dalam penyelesaian tugas akhir ini, penulis telah mendapat begitu banyak bantuan baik moral maupun materiil dari banyak pihak. Atas bantuan yang telah diberikan, penulis ingin menyampaikan penghargaan dan ucapan terima kasih yang sedalam-dalamnya kepada:

1. Wayan Firdaus Mahmudy, SSi., MT selaku pembimbing I sekaligus Ketua Program Studi Ilmu Komputer dan Dian Eka Ratnawati, SSi., M.Kom selaku pembimbing II. Terima kasih atas semua saran, kritik, waktu, bantuan, dorongan semangat dan bimbingannya.
2. Drs. Muh. Arif Rahman, M.Kom selaku Penasihat Akademik.
3. Dr. Agus Suryanto, MSc selaku Ketua Jurusan Matematika.
4. Segenap bapak dan ibu dosen yang telah mendidik dan mengajarkan ilmunya kepada penulis.
5. Segenap staf dan karyawan di Jurusan Matematika FMIPA Universitas Brawijaya
6. Bapak, Ibu dan adik. Terima kasih atas cinta, kasih sayang, doa, dukungan dan semangat yang tiada henti.
7. Sahabat-sahabat ilkom angkatan 2003-2005, terima kasih atas semua bantuan yang telah diberikan.
8. Teman-teman kos Kertoasri 93, terima kasih atas dukungannya.
9. Pihak lain yang telah membantu terselesaikannnya Tugas Akhir ini yang tidak bisa penulis sebutkan satu-persatu.

Semoga penulisan laporan tugas akhir ini bermanfaat bagi pembaca sekalian. Sebagai manusia biasa yang hanya terbiasa dengan hal-hal biasa, penulis menyadari bahwa tugas akhir ini masih jauh dari kesempurnaan, dan mengandung banyak kpekurangan, sehingga dengan segala kerendahan hati penulis mengharapkan kritik dan saran yang membangun dari pembaca.

Malang, Juni 2007

Penulis

DAFTAR ISI

Halaman

BAB I
PENDAHULUAN

1.1 Latar Belakang

Penjadwalan adalah salah satu elemen perencanaan dan pengendalian produksi. Penjadwalan produksi merupakan komponen yang sangat penting dalam sebuah proses produksi karena dengan penjadwalan produksi yang optimal diharapkan dapat memperlancar proses produksi. Untuk memaksimalkan penggunaan keseluruhan mesin dan meminimalkan biaya produksi, maka perlu dilakukan penjadwalan kerja yang tepat bagi setiap mesin. Penjadwalan ini harus memperhatikan faktor jumlah mesin dan faktor banyaknya pekerjaan yang harus diselesaikan. Dengan menempatkan susunan atau urutan pekerjaan yang tepat pada setiap mesin untuk mengurangi waktu menganggur pada mesin-mesin yang tersedia, maka secara tidak langsung akan meningkatkan produktivitas serta meminimumkan biaya produksi.[Limyana,2002]

Masalah penjadwalan *flow shop* adalah menjadwalkan proses produksi dari masing-masing n *job* yang mempunyai urutan proses produksi dan melalui m mesin yang sama dengan tujuan untuk menentukan urutan dari *job* yang ada agar dapat memberikan performance yang optimal.[Medianti,1999]

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*. Metode *heuristik* menggunakan suatu algoritma tertentu yang mencari solusi terdekat dengan solusi optimal tanpa melakukan pengujian pada semua kemungkinan solusi yang ada. Hasil yang didapat memang tidak mungkin paling optimal tetapi hampir optimal. [Onwubolu,1999]

Dalam perkembangan perindustrian, banyak literatur menganjurkan penggunaan prosedur metaheuristik untuk optimalisasi. Sebuah komite yang menangani riset dan operasi menyelidiki beberapa diantaranya, yaitu *Simulated Annealing*, *Tabu Search*, *Target Analysis* dan *Neural Network*, dan disimpulkan bahwa metode-metode ini memiliki potensi yang besar untuk menyelesaikan masalah optimalisasi. Beberapa tahun kemudian pengujian metode baru yaitu algoritma genetika mulai dilakukan dan disimpulkan bahwa algoritma genetika dapat digunakan sebagai teknik multisolusi, tidak memerlukan penyesuaian untuk masalah-masalah apapun dan solusi yang dihasilkan mendekati optimal.[Onwubolu,1999].

Penelitian-penelitian sebelumnya dalam hal penjadwalan biasanya difokuskan pada pemenuhan deadline [Medianti,1999]. Padahal ada aspek lain yang cukup penting seperti meminimumkan total *flowtime* atau meminimumkan *makespan* yang lebih efektif dalam mengurangi biaya penjadwalan.[French,1982]. Oleh karena itu penelitian ini menggunakan dua kriteria yaitu *makespan* dan total *flowtime* dengan tujuan untuk berusaha melihat kemampuan dua algoritma (Genetika dan *Simulated Annealing*) dalam menyelesaikan masalah penjadwalan *flow shop*.

1.2 Rumusan Masalah

Rumusan masalah dalam tugas akhir ini adalah:

1. Bagaimana mengimplementasikan penjadwalan *flow shop* (*flow shop scheduling*) menggunakan algoritma *simulated annealing* dan algoritma genetika?
2. Membandingkan hasil yang diperoleh dari uji coba implementasi dua algoritma baik algoritma *simulated annealing* maupun algoritma genetika dalam *flow shop scheduling*.

1.3 Batasan Masalah

Batasan masalah dalam penulisan tugas akhir ini adalah:

1. Algoritma yang diperbandingkan kinerjanya adalah algoritma genetika dan *simulated annealing* sebagai dua algoritma *metaheuristik*.
2. Pembuatan desain dan program komputer yang dilakukan hanya menggunakan algoritma genetika dan *simulated annealing* saja.
3. Sistem produksi adalah *flow shop*.
4. Proses produksi dari *job-job* sudah diketahui secara jelas.
5. Setiap *job* memiliki ready time yang sama.
6. Kriteria yang digunakan adalah *makespan* dan total *flowtime*.
7. Tiap algoritma yang diperbandingkan memiliki jumlah iterasi yang sama (masing-masing algoritma mempunyai kesempatan yang sama dalam mencari solusi akhir).

1.4 Tujuan

Tujuan yang ingin dicapai dari pembuatan tugas akhir ini adalah:

1. Mengetahui prosentase keunggulan diantara dua algoritma yang diperbandingkan (keunggulan algoritma genetika atas algoritma *simulated annealing* atau sebaliknya, keunggulan algoritma *simulated annealing* terhadap algoritma genetika) dengan kriteria *makespan* dan *total flow time*. Algoritma dapat disebut unggul jika algoritma tersebut memiliki nilai *makespan* dan total *flowtime* yang lebih kecil dibandingkan nilai *makespan* dan total *flowtime* yang dihasilkan algoritma lain.
2. Melakukan analisis terhadap hasil implementasi program yang diperoleh, yaitu perbandingan kinerja dari algoritma genetika dan algoritma *simulated annealing* dengan kriteria *makespan* dan total *flowtime*.

1.5 Manfaat

Manfaat yang diperoleh dari penulisan tugas akhir ini adalah memberikan gambaran mengenai *makespan* dan total *flowtime* yang diperoleh dari implementasi *flow shop scheduling* menggunakan metode *algoritma simulated annealing* dan algoritma genetika dengan spesifikasi perangkat keras yang digunakan dalam pengerjaan tugas akhir ini.

BAB II
TINJAUAN PUSTAKA

2.1 Penjadwalan

2.1.1. Pendahuluan

Penjadwalan didefinisikan sebagai proses pengambilan keputusan, dimana melibatkan beragam sumber daya yang tersedia secara terbatas untuk menyelesaikan tugas-tugas dalam jangka waktu tertentu, dimana pada intinya digunakan untuk memperoleh efisiensi penggunaan fasilitas, waktu dan untuk meminimumkan biaya.[Baker,1994].

Masalah penjadwalan dapat digambarkan sebagai berikut : [Baker,1994].

a. Menggambarkan lingkungan mesin, yang terdiri dari satu masukan. Contoh lingkungan mesin :
   1) *Single machine* (1), terdapat hanya 1 mesin dalam melakukan proses produksi untuk mendapatkan produk akhir.
   2) *Identical machine in paralel* (P_m), terdapat m mesin identik dalam susunan parallel.
   3) *Flow shop* (F_o), terdapat *m* mesin yang tersusun secara seri.
   4) *Job shop* (J_m), setiap *job* memiliki rute proses sendiri.
b. Menggambarkan rincian karakteristik proses terdiri dari nol, satu atau beberapa masukan. Beberapa karakteristik proses yaitu :
   1) *Release dates* (r_j), *job* ke-j tidak akan memulai proses sebelum *release dates*-nya.
   2) *Pre-emption* (prmp), menyatakan bahwa *job* yang sedang diproses dalam sebuah mesin diperbolehkan untuk disela dengan *job* yang lain meskipun *job* itu belum selesai.
   3) Menggambarkan tujuan yang akan diminimumkan, terdiri dari satu masukan.
c. Faktor-faktor yang menggambarkan karakteristik, antara lain :
   1) Jumlah pekerjaan (*job*) yang dijadwalkan, yaitu jumlah *job* yang akan diproses, waktu yang diperlukan untuk tiap proses dan jenis mesin yang dibutuhkan.
   2) Jumlah mesin dan operasi pada *workshop*.
   3) Pola fasilitas manufaktur : *flow shop*, *job shop*, *open shop*.
   4) Pola kedatangan pekerjaan pada fasilitas manufaktur : statis (*ready time* = 0) dan dinamis.

2.1.2. Lingkup Penjadwalan

Terdapat cukup banyak lingkungan penjadwalan, tabel di bawah ini akan menjelaskan lingkungan penjadwalan tersebut [Morton, 1993].

Tabel 2.1 Lingkungan Penjadwalan (Morton,1993)

Type	Karakteristik
Classic Job Shop	Diskrit, aliran rumit, *job* unik, no multi-use parts
Open Job Shop	Diskrit, aliran rumit, ada *job* yang berulang, dan/atau multi-use parts
Batch Shop	Diskrit, aliran tidak begitu rumit, banyak pengulangan dan multi-use parts
Flow Shop	Diskrit, aliran linear, *job*-*job* hampir sama
Assembly Shop	Bentuk perakitan dari *job shop* atau *batch shop*

Karena dalam skripsi ini hanya berkaitan dengan lingkungan *flow shop*,maka hanya *flow shop scheduling* saja yang dibahas lebih lanjut.

2.1.3. Tujuan Penjadwalan

Beberapa macam tujuan penjadwalan antara lain : [Baker, 1994]

- *Makespan* (Cmax)
  Merupakan waktu yang diperlukan untuk menyelesaikan semua operasi (tugas) dalam suatu lintasan produksi yaitu dari *job* pertama masuk untuk diproses sampai *job* terakhir selesai diproses.
- *Total Flowtime* (Ft)
  Merupakan total waktu semua mesin bekerja untuk semua *job* yang ada.
- *Tardiness* (Tj)
  Merupakan tujuan untuk memperoleh waktu keterlambatan paling minimum dari seluruh *job* yang diproses.
- *Lateness* (Lmax).
  Tujuan untuk mencapai jumlah waktu keterlambatan yang minimum.

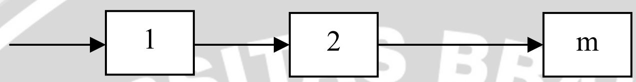
2.1.4. Flow Shop

Pada skripsi ini akan dibahas mengenai penjadwalan flow shop. *Flow shop* pada dasarnya adalah batch shop dengan aliran yang linear. Penjadwalan *flow shop* dicirikan oleh adanya aliran kerja yang satu arah dan teratur. Pada penjadwalan ini terdapat beberapa mesin yang disusun secara seri dan setiap *job* yang ada harus melalui semua mesin dengan urutan mesin yang sama di dalam prosesnya.[Morton,1993].

Ada beberapa macam pola *flow shop*, yaitu : [Morton,1993].

1) *Pure flow shop*

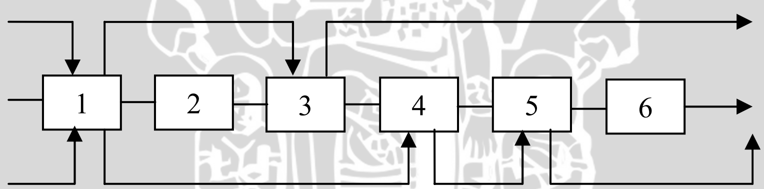
   Dalam *pure flow shop*, semua *job* memerlukan satu operasi pada tiap mesin dan dapat juga dikatakan sebagai *simple flow shop* karena tiap *job* memerlukan jumlah mesin dan urutan yang sama.



Gambar 2.1 Pola *Pure Flow Shop*

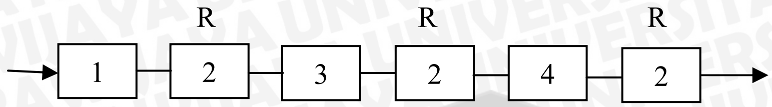
2) *Skip flow shop*

   Dalam general *flow shop*, semua *job* mungkin membutuhkan kurang dari m operasi dan operasi-operasi tersebut mungkin tidak memerlukan pasangan mesin sesuai urutan nomornya dan mungkin tidak selalu berawal atau berakhir pada mesin yang sama. Bagi *job* yang jumlah operasinya kurang dari m, waktu pemrosesan untuk operasi yang bersangkutan diberi nilai nol. Pola ini disebut juga *skip flow shop* atau pola melompat, dimana mesin-mesin tertentu dapat dilompati oleh *job-job* tertentu.



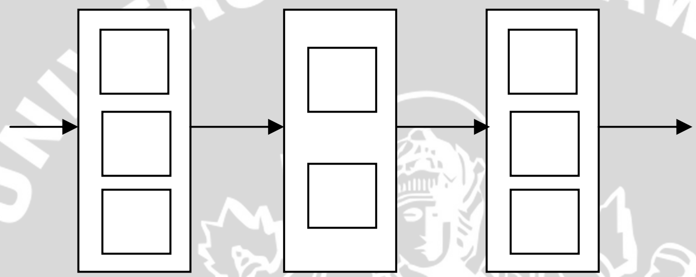
Gambar 2.2 Pola *Skip Flow Shop*

3) *Re-entrant flow shop*
   Dalam *re-entrant flow shop*, beberapa mesin dapat dilalui lebih dari sekali. Hal ini sulit dimodelkan secara analitis tetapi relatif mudah bila digunakan pendekatan heuristic seperti *shifting bottleneck model* atau *bottleneck dynamis*.



Gambar 2.3 Pola *Re-entrant Flow Shop*

4) *Compound flow shop*
   Dalam *compound flow shop*, satu mesin dalam set mesin dapat digantikan oleh sekelompok mesin. Kelompok mesin tersebut umumnya adalah mesin-mesin paralel atau jalur *batch* yang diikuti oleh mesin-mesin paralel.



Gambar 2.4 Pola *Compound Flow Shop*

2.1.5. Notasi

Beberapa notasi yang biasa digunakan dalam penjadwalan antara lain [Morton,1993] :

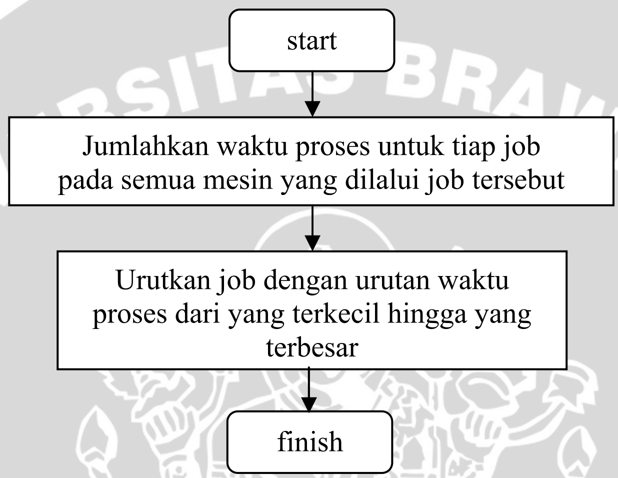
- n; menyatakan jumlah *job*.
- m; menyatakan jumlah mesin.
- *subscript* i; menyatakan mesin ke-i
- *subscript* j; menyatakan *job* ke-j
- Pij; menyatakan waktu proses *job* ke-j pada mesin ke-i.

- Rij; menyatakan waktu *job* ke-j tiba pada suatu sistem atau waktu tercepat *job* ke-j dimulai prosesnya.
- Dj; menyatakan Due Date yaitu waktu penyelesaian yang dijanjikan.
- Wj; menyatakan faktor prioritas yang tergantung pada kepentingan relatif suatu *job* ke-j terhadap *job* lainnya.

2.2 Algoritma SPT (Shortest Processing Time)

2.2.1. Pengertian

Algoritma SPT (*Shortest Processing Time*) termasuk algoritma yang sangat sederhana. Inti dari algoritma SPT adalah mencari waktu proses *job* yang terkecil dan mengurutkan hingga yang terbesar. Secara skematik akan lebih mudah dijabarkan sebagai berikut :



Gambar 2.5 Flowchart Algoritma *SPT* (*Shortest Processing Time*) [Medianti,1999].

2.2.2. Contoh Algoritma SPT (*Shortest Processing Time*)

Agar lebih jelas bagaimana perhitungan dari algoritma ini diberikan contoh manual dari kasus yang sederhana seperti di bawah ini

Tabel 2.2 Problem Penjadwalan Sederhana

Job	Mesin (per satuan waktu)				
	1	**2**	**3**	**4**	**5**
1	2	1	8	2	2
2	6	3	2	3	4
3	1	4	3	8	1
4	4	9	3	7	3
5	6	8	7	3	3

Dengan algoritma SPT (*Shortest Processing Time*), dibuat jadwal pertama sebagai berikut :

Tabel 2.3 Problem sederhana dengan total waktu *job*

Job	Mesin (per satuan waktu)					total
	1	**2**	**3**	**4**	**5**	
1	2	1	8	2	2	15
2	6	3	2	3	4	18
3	1	4	3	8	1	17
4	4	9	3	7	3	26
5	6	8	7	3	3	27

Kemudian *job-job* tersebut diurutkan berdasarkan urutan total waktu terkecil hingga total waktu terbesar, maka akan didapatkan hasil sebagai berikut : 1-3-2-4-5.

2.3 Formulasi *Time Table* pada *Flow Shop Scheduling*

Umumnya pada sistem produksi yang bersifat *flow shop* terdiri dari beberapa mesin (m) dan mempunyai sejumlah *job* yang harus dikerjakan (n) serta waktu proses per unit *job* i pada mesin j, t_{ij} (untuk i = 1,…,n; j = 1,…,m), maka :

$$T_{ij} = t_{ij} \times d_i$$

dengan d_i adalah jumlah permintaan untuk *job* i dan T_{ij} adalah total waktu proses (sesuai *demand*) *job* i pada mesin j serta saat dimulainya *job* i pada mesin j (S_{ij}) dan saat selesainya *job* i pada mesin j (E_{ij}). Sedangkan t_{ij} adalah waktu proses per unit *job* posisi ke-i pada mesin j dan T_{ij} merupakan total waktu proses *job* ke-i (dalam sequence) pada mesin j, maka saat dimulainya job urutan ke-i pada mesin j (S_{ij}) dan saat selesainya *job* ke-i pada mesin j (E_{ij}) dapat dirumuskan : [Medianti,1999].

- Untuk j = 1
  - ➔ $S_{[1]\,1} = 0$ $\qquad E_{[1]\,1} = T_{[1]\,1}$
  - ➔ $S_{[i]\,1} = E_{[i-1]\,1}$ $\qquad E_{[i]\,1} = S_{[i]\,1} + T_{[1]\,1}$
- Untuk j = 2
  - ➔ $S_{[1]\,2} = E_{[1]\,1}$ $\qquad E_{[1]\,2} = S_{[1]\,2} + T_{[1]\,2}$

  Jika $E_{[i-1]\,2} \leq E_{[i]\,1}$ maka
  - ➔ $S_{[i]\,2} = E_{[i]\,1}$ $\qquad E_{[i]\,2} = S_{[i]\,2} + T_{[i]\,2}$

  Jika $E_{[i-1]\,2} > E_{[i]\,1}$ maka
  - ➔ $S_{[i]\,2} = E_{[i-1]\,2}$ $\qquad E_{[i]\,2} = S_{[i]\,2} + T_{[i]\,2}$, ([i] = 2,3,…,n)
- Formulasi untuk j = 3,4,…,m sama dengan formulasi j = 2.

Sebagai contoh dapat digambarkan sebagai berikut :

Tabel 2.4 Problem penjadwalan
(waktu yang dibutuhkan tiap *job*)

Job	**Mesin (per satuan waktu)**				
	1	**2**	**3**	**4**	**5**
1	2	1	8	2	2
2	6	3	2	3	4
3	1	4	3	8	1
4	4	9	3	7	3
5	6	8	7	3	3

Setelah diurutkan berdasarkan total waktu proses, maka didapatkan *initial time table* seperti terlihat pada tabel 2.5 berikut ini :

Tabel 2.5 *Time table* pada *flow shop* (per satuan waktu)

Job	Mesin 1		Mesin 2		Mesin 3		Mesin 4		Mesin 5	
	S	**E**	**S**	**E**	**S**	**E**	**S**	**E**	**S**	**E**
1	0	2	2	3	3	11	11	13	13	15
3	2	3	3	7	11	14	14	22	22	23
2	3	9	9	12	14	16	22	25	25	29
4	9	13	13	22	22	25	25	32	32	35
5	13	19	22	30	30	37	37	40	40	43

Keterangan dari tabel 2.5 di atas :

S : Start /saat dimulainya job urutan ke-i pada mesin ke-j (S_{ij}).
E : End / saat selesainya *job* ke-i pada mesin ke-j (E_{ij}).

Hasil *time table* tersebut didapat dari proses berikut ini :

- Proses *job* ke-1 mesin ke-1
  S = 0, E = 2; karena proses baru dimulai, maka nilai S = 0 dan nilai E didapat dari nilai S (0) ditambah T (waktu pada *job* 1 mesin 1) → 0 + 2 = 2.
- Proses *job* ke-1 mesin ke-2
  S = 2, E = 3; nilai S = 2 didapat dari nilai E sebelumnya dan nilai E didapat dari nilai S (2) ditambah T (waktu pada *job* 1 mesin 2) → 2 + 1 = 3.
- Proses *job* ke-1 mesin ke-3
  S = 3, E = 11; nilai S = 3 didapat dari nilai E sebelumnya dan nilai E didapat dari nilai S (3) ditambah T (waktu pada *job* 1 mesin 3) → 3 + 8 = 11.
- Proses *job* ke-1 mesin ke-4
  S = 11, E = 13; nilai S = 11 didapat dari nilai E sebelumnya dan nilai E didapat dari nilai S (11) ditambah T (waktu pada *job* 1 mesin 4) → 11 + 2 = 13.
- Proses *job* ke-1 mesin ke-5
  S = 13, E = 15; nilai S = 13 didapat dari nilai E sebelumnya dan nilai E didapat dari nilai S (13) ditambah T (waktu pada job 1 mesin 5) → 13 + 2 = 15.

- Proses *job* ke-3 mesin ke-1
  S = 2, E = 3; nilai S = 2 didapat dari nilai E pada *job* 1 mesin 1 dan nilai E didapat dari nilai S (2) ditambah T (waktu pada *job* 3 mesin 1) → 2 + 1 = 3.
- Proses *job* ke-3 mesin ke-2
  S = 3, E = 7; nilai S = 3 didapat dari perbandingan nilai E *job* sebelumnya (3) dengan nilai E pada mesin sebelumnya (3) dan diambil nilai yang paling besar (dalam hal ini tidak ada yang paling besar) dan nilai E didapat dari nilai S (3) ditambah T (waktu pada *job* 3 mesin 2) → 3 + 4 = 7.
- Proses *job* ke-3 mesin ke-3
  S = 11, E = 14; nilai S = 11 didapat dari perbandingan nilai E *job* sebelumnya (7) dengan nilai E pada mesin sebelumnya (11) dan diambil nilai yang paling besar (dalam hal ini nilai E pada mesin sebelumnya yang paling besar) dan nilai E didapat dari nilai S (11) ditambah T (waktu pada *job* 3 mesin 3) → 11 + 3 = 14.

Begitu seterusnya,lakukan langkah yang sama hingga akhir (*job* ke-5, mesin ke-5).

2.4 *Makespan* dan Total *Flowtime*

Menurut Johnson, *makespan* adalah waktu yang diperlukan untuk menyelesaikan semua operasi (tugas) dalam suatu lintasan produksi yaitu dari *job* pertama masuk untuk diproses sampai *job* terakhir selesai diproses. Sedangkan total *flowtime* merupakan total waktu semua mesin bekerja untuk semua *job* yang ada.[Baker,1994]. Sebagai contoh dapat dilihat pada tabel 2.6 berikut ini :

Tabel 2.6 *Makespan* dan Total *Flowtime* (per satuan waktu)

Job	**Mesin 1**		**Mesin 2**		**Mesin 3**		**Mesin 4**		**Mesin 5**	
	S	**E**	**S**	**E**	**S**	**E**	**S**	**E**	**S**	**E**
1	0	2	2	3	3	11	11	13	13	15
3	2	3	3	7	11	14	14	22	22	23
2	3	9	9	12	14	16	22	25	25	29
4	9	13	13	22	22	25	25	32	32	35
5	13	19	22	30	30	37	37	40	40	**43**

Nilai *makespan* pada tabel 2.6 di atas adalah 43 dan total *flow time* tabel di atas adalah jumlah total waktu proses masing-masing *job* yaitu 15 + 23 + 29 + 35 + 43 = 145.

2.5 *Simulated Annealing*

2.5.1. Pengertian

Simulated annealing dikembangkan berdasarkan ide dari mekanisme perilaku pendinginan dan proses kristalisasi (*annealing*) material panas. Algoritma ini melakukan peningkatan iteratif untuk memperbaiki solusi yang dihasilkan teknik-teknik penjadwalan heuristik, dalam hal ini adalah sebuah solusi awal yang dibuat dengan teknik heuristik atau random.[Cheng,1999].

Proses *annealing* akan diawali dengan material yang panas yang kemudian temperaturnya diturunkan secara bertahap hingga keadaan energi terendah (*ground state*) didapatkan. Apabila pendinginan dilakukan terlalu cepat atau apabila temperatur awalnya terlalu rendah, maka benda padat yang dihasilkan akan cenderung berada di tingkat energi yang lebih tinggi (*metastable state*) daripada *ground state*. Pada kondisi ini, benda padat itu akan lebih rentan terhadap kemungkinan patah dan retak. [Cheng,1999].

Algoritma *simulated annealing* diawali dengan sebuah solusi awal yang didapatkan dengan menggunakan algoritma heuristik lain yang lebih sederhana seperti *Shortest Processing Time*, *Longest Processing Time* dan lain-lain. Kemudian solusi awal ini dihitung nilai objektifnya. Langkah selanjutnya adalah mencari *neighborhoods* berikutnya dengan metode tertentu ataupun random. Dihitung pula nilai objektif solusi ini dan dibandingkan dengan solusi sebelumnya. Jika solusi kedua memiliki nilai objektif yang lebih baik maka solusi itu diterima dan menjadi solusi awal bagi pencarian *neighborhoods* berikutnya. Namun jika tidak, solusi itu masih bisa diterima dengan probabilitas tertentu sesuai dengan algoritma *simulated annealing*. Dengan demikian satu iterasi telah diselesaikan. [Ponnambalan,1999].

Setelah melalui beberapa *replikasi* (pengubahan / pencarian solusi *neighborhoods* dalam satu tingkat suhu yang sama), suhu dapat diturunkan sesuai dengan parameter yang ditentukan. Algoritma dihentikan apabila sejumlah iterasi yang diharapkan telah selesai dilakukan, apabila suatu tingkat suhu tertentu telah dicapai atau apabila tidak ada lagi perubahan nilai objektif untuk beberapa iterasi berturut-turut (*frozen*). [Ponnambalan,1999].

2.5.2. Skema

Struktur algoritma *simulated annealing* secara umum adalah sebagai berikut : [Ponnambalan,1999].

1. Cari solusi awal S menggunakan parameter awal dan metode heuristik awal yang dapat ditentukan sendiri.
2. Tetapkan suatu nilai temperatur awal T_0 yang cukup tinggi, dimana $T_0 > 0$.
3. Pada keadaan tidak *frozen*, lakukan :
   a. Lakukan L kali :
      i. Cari solusi *neighborhood* Sb dari Sa menggunakan metode yang dapat ditetapkan sendiri.
      ii. Δ = nilai objektif (Sb) - nilai objektif (Sa).
      iii. Jika Δ < 0, maka tetapkan Sa = Sb. Jika tidak, maka tetapkan Sa = Sb dengan probabilitas exp(-Δ/T).
   b. T = r x T, dimana r adalah faktor reduksi suhu.
4. Dapatkan solusi optimal.

Adapun perhitungannya dapat dilihat dari beberapa rumusan berikut ini :

- Untuk menghitung nilai Δ dapat dilakukan dengan rumus berikut

$$e1 = w1 \left(\frac{(Ms - \min(Ms, Ms\ new))}{\min(Ms, Ms\ new)}\right) + w2 \left(\frac{(F - \min(F, F\ new))}{\min(F, F\ new)}\right) \quad (2.1)$$

$$e2 = w1 \left(\frac{(Ms\ new - \min(Ms, Ms\ new))}{\min(Ms, Ms\ new)}\right) + w2 \left(\frac{(F\ new - \min(F, F\ new))}{\min(F, F\ new)}\right) \quad (2.2)$$

$$\Delta = e2 - e1 \quad (2.3)$$

Keterangan :

w1 : bobot *makespan*
w2 : bobot *flowtime*
Ms : nilai *makespan* awal
Ms new : nilai *makespan* baru
F : nilai *flowtime* awal
F new : nilai *flowtime* baru
e1 : pembobotan *makespan* dan total *flowtime* awal terhadap *makespan* dan total *flowtime* baru
e2 : pembobotan *makespan* dan total *flowtime* baru terhadap *makespan* dan total *flowtime* awal

Untuk bobot, jumlah w1 dan w2 sama dengan 1 (misal nilai w1 dan w2 adalah 0,5 maka bobot kepentingannya sama).

- Untuk menghitung nilai probabilitas (p) dengan rumusan
  $p = \exp\left(-\frac{\Delta}{T0}\right)$, dimana T_0 adalah suhu awal (2.4)

- Untuk menghitung hasil reduksi suhu dengan rumusan

  $T_i = r \times T_{i-1}$, dimana T adalah suhu dan r adalah faktor reduksi (2.5)

2.5.3. Notasi dan Istilah

Notasi dan istilah yang akan digunakan pada algoritma *simulated annealing* antara lain : [Ponnambalan,1999]

1. *Parameter awal*
   Parameter awal ini ditentukan sendiri dan memegang peranan yang sangat penting karena akan mempengaruhi jalannya algoritma. Parameter awal ini terdiri dari :
   a. Temperatur awal (T_0), merupakan penanda awal iterasi dimana temperatur awal ini akan terus berkurang hingga mencapai temperatur akhir.
   b. Temperatur akhir (Ta), merupakan batas akhir dari iterasi dimana iterasi sudah dapat dihentikan.

c. Faktor reduksi suhu (r), merupakan angka yang digunakan untuk menurunkan suhu secara bertahap.
d. Angka replikasi (R), merupakan angka yang menunjukkan berapa kali loop (perulangan) yang harus dilakukan sebelum menurunkan suhu.
e. Bobot (w), merupakan nilai bobot untuk masing-masing kepentingan (*makespan* dan total *flowtime*).

2. *Solusi awal*

   Solusi awal adalah suatu jadwal yang menjadi acuan awal dimulainya algoritma *simulated annealing*. Untuk mendapatkan solusi awal tidak diperlukan algoritma yang terlalu rumit karena diharapkan solusi awal ini akan dibuang dan digantikan oleh solusi lain yang jauh lebih optimal. Dalam hal ini dapat dilakukan dengan menggunakan algoritma *Shortest Processing Time* (*SPT*).

3. *Tetangga* (*neighborhood*)

   Tetangga (*neighborhood solution*) adalah jadwal atau solusi yang didapatkan dari solusi sebelumnya dengan metode tertentu. Cara pembuatan jadwal *neighborhood* ini tidak ada teorinya sehingga harus ditentukan sendiri suatu metode yang akan digunakan. Contoh metode yang sudah ada adalah metode pertukaran *job* secara random atau pertukaran semua *job* dengan batasan yang random.

4. *Probabilitas*

   Ciri dari algoritma *simulated annealing* adalah dimungkinkannya penerimaan solusi yang lebih jelek daripada solusi sebelumnya asalkan dapat memenuhi suatu probabilitas tertentu. Hal inilah yang membuat solusi tidak terjebak dalam optimum lokal. Probabilitas yang diberikan adalah suatu probabilitas eksponensial dengan rumus $exp(-\Delta/T)$. Δ adalah selisih nilai objektif dari solusi yang baru dengan solusi sebelumnya, sedangkan T adalah tingkat suhu pada saat iterasi tersebut terjadi. Maksud dari rumus ini adalah diharapkan pada suhu yang cukup tinggi, probabilitas penerimaan solusi yang jelek adalah cukup besar, namun seiring dengan turunnya suhu, probabilitas penerimaan solusi yang jelek ini juga menurun, hingga ketika mendekati suhu akhir probabilitas penerimaannya akan sangat kecil.

2.5.4. Contoh Manual

Pada contoh kasus kali ini, akan digunakan permasalahan seperti tabel 2.2 yang telah didapatkan solusi awalnya dengan algoritma *Shortest Processing Time* (SPT).

- **Parameter awal** :
  - Suhu awal (T_0) = 0,50
  - Suhu akhir (Ta) = 0,23
  - Faktor reduksi suhu (r) = 0,7
  - Replikasi (R) = 2
  - w1 = w2 = 0,5 (karena bobot kepentingannya sama)
    Yang dimaksud dengan bobot kepentingan yang sama disini adalah bahwa tidak ada kriteria yang dititikberatkan antara *makespan* dan total *flowtime* yang dicari.

- **Problem awal**

Tabel 2.7 Problem Awal

Job	**Mesin (per satuan waktu)**				
	1	**2**	**3**	**4**	**5**
1	2	1	8	2	2
2	6	3	2	3	4
3	1	4	3	8	1
4	4	9	3	7	3
5	6	8	7	3	3

Problem ini sudah diketahui nilai total waktu *job*-nya dengan menggunakan algoritma *Shortest Processing Time* (*SPT*) dan yang terbaik yaitu 1-3-2-4-5. Jadwal ini menjadi sebuah solusi awal bagi algoritma *simulated annealing* dan dapat dilanjutkan dengan mancari nilai objektifnya.

- **Penjadwalan Solusi Awal**

Tabel 2.8 Penjadwalan Solusi Awal (per satuan waktu)

Job	Mesin 1		Mesin 2		Mesin 3		Mesin 4		Mesin 5	
	S	E	S	E	S	E	S	E	S	E
1	0	2	2	3	3	11	11	13	13	15
3	2	3	3	7	11	14	14	22	22	23
2	3	9	9	12	14	16	22	25	25	29
4	9	13	13	22	22	25	25	32	32	35
5	13	19	22	30	30	37	37	40	40	43

Dari tabel 2.8, maka didapatkan nilai objektif *makespan* (Ms) sebesar 43 dan total *flowtime* (F) sebesar 145.

- **Iterasi 1.1**

  T_0 = 0,5; R = 1; Sa = 1-3-2-4-5;
  makespan (Ms) = 43; total *flowtime* (F) = 145.
  Mencari 2 buah nilai random r1 dan r2 antara 1 hingga 5 (sejumlah *job*). Misal didapatkan r1 = 2 dan r2 = 3, maka tukarkan posisi *job* ke-2 dan *job* ke-3.
  Sb : 1-2-3-4-5
  Penjadwalan iterasi 1.1

Tabel 2.9 Penjadwalan Iterasi 1.1 (per satuan waktu)

Job	Mesin 1		Mesin 2		Mesin 3		Mesin 4		Mesin 5	
	S	E	S	E	S	E	S	E	S	E
1	0	2	2	3	3	11	11	13	13	15
2	2	8	8	11	11	13	13	16	16	20
3	8	9	11	15	15	18	18	26	26	27
4	9	13	15	24	24	27	27	34	34	37
5	13	19	24	32	32	39	39	42	42	45

didapatkan *makespan* (Ms) sebesar 45 dengan total *flowtime* (F) sebesar 144.
Dihtung besar e1 (sesuai rumus 2.1), e2 (sesuai rumus 2.2) dan Δ (sesuai rumus 2.3) untuk jadwal 1.1 sebagai berikut :
Ms = 43; Ms new = 45; F = 145; F new = 144; maka

$$e1 = 0{,}5\left(\frac{(43-\min(43{,}45))}{\min(43{,}45)}\right) + 0{,}5\left(\frac{(145-\min(145{,}144))}{\min(145{,}144)}\right)$$
$$= 0{,}0034$$

$$e2 = 0{,}5\left(\frac{(45-\min(43{,}45))}{\min(43{,}45)}\right) + 0{,}5\left(\frac{(144-\min(145{,}144))}{\min(145{,}144)}\right)$$

$$= 0{,}5\left(\frac{2}{43}\right) + 0 = 0{,}023 + 0 = 0{,}023$$

$\Delta = e2 - e1 = 0{,}023 - 0{,}0034 = 0{,}0196$

Karena $\Delta \geq 0$ yang menandakan bahwa jadwal baru tidak lebih baik dari jadwal lama, maka dihitung *probabilitas* (sesuai rumus 2.4) penerimaan sebagai berikut :

$p = \exp(-\frac{0{,}0196}{0{,}5}) = \exp(-0{,}0392) = 0{,}938$

diambil angka random antara 0 hingga 1, misal didapat px = 0,356. Karena px $\leq$ p, maka jadwal baru tersebut bisa diterima.

- **Iterasi 1.2**

  T_0 = 0,5; R = 2; Sa = 1-2-3-4-5;
  Makespan (Ms) = 45; total *flowtime* (F) = 144
  Mencari 2 buah nilai random r1 dan r2 antara 1 hingga 5 (sejumlah *job*). Misal didapatkan r1 = 4 dan r2 = 2, maka tukarkan posisi *job* ke-4 dan *job* ke-2.
  Sb : 1-4-3-2-5

  Penjadwalan iterasi 1.2

Tabel 2.10 Penjadwalan Iterasi 1.2 (per satuan waktu)

Job	Mesin 1		Mesin 2		Mesin 3		Mesin 4		Mesin 5	
	S	E	S	E	S	E	S	E	S	E
1	0	2	2	3	3	11	11	13	13	15
4	2	6	6	15	15	18	18	25	25	28
3	6	7	15	19	19	22	25	33	33	34
2	7	13	19	22	22	24	33	36	36	40
5	13	19	22	30	30	37	37	40	40	43

didapatkan *makespan* (Ms) sebesar 43 dengan total *flowtime* (F) sebesar 160.
Dihtung besar e1 (sesuai rumus 2.1), e2 (sesuai rumus 2.2) dan Δ (sesuai rumus 2.3) untuk jadwal 1.2 sebagai berikut :

Ms = 45; Ms new = 43; F = 144; F new = 160; maka

$$e1 = 0{,}5\left(\frac{(45-\min(45{,}43))}{\min(45{,}43)}\right) + 0{,}5\left(\frac{(144-\min(144{,}160))}{\min(144{,}160)}\right)$$

$= 0{,}0232 + 0$

$$e2 = 0{,}5\left(\frac{(43-\min(45{,}43))}{\min(45{,}43)}\right) + 0{,}5\left(\frac{(160-\min(144{,}160))}{\min(144{,}160)}\right)$$

$= 0 + 0{,}0555 = 0{,}0555$
$\Delta = e2 - e1 = 0{,}0555 - 0{,}0232 = 0{,}0323.$

Karena $\Delta \geq 0$ yang menandakan bahwa jadwal baru tidak lebih baik dari jadwal lama, maka dihitung *probabilitas* (sesuai rumus 2.4) penerimaan sebagai berikut :

$p = \exp\left(-\frac{0{,}0323}{0{,}5}\right) = \exp(-0{,}0646) = 0{,}9374$

diambil angka random antara 0 hingga 1, misal didapat px = 0.674. Karena px ≤ p, maka jadwal baru tersebut bisa diterima.
Pada parameter awal telah ditentukan nilai *replikasi* sebesar 2 dan saat ini nilai R = 2, maka suhu harus diturunkan sesuai dengan faktor reduksi (sesuai rumus 2.5) yang telah ditentukan.

- **Iterasi 2.1**
  $T_0 = 0{,}5 \times 0{,}7 = 0{,}35$; R = 1; Sa = 1-4-3-2-5;
  Makespan (Ms) = 43; total *flowtime* (F) = 160
  Mencari 2 buah nilai random r1 dan r2 antara 1 hingga 5 (sejumlah *job*). Misal didapatkan r1 = 1 dan r2 = 4, maka tukarkan posisi *job* ke-1 dan *job* ke-4.
  Sb : 2-4-3-1-5

Penjadwalan iterasi 2.1

Tabel 2.11 Penjadwalan Iterasi 2.1 (per satuan waktu)

Job	Mesin 1		Mesin 2		Mesin 3		Mesin 4		Mesin 5	
	S	E	S	E	S	E	S	E	S	E
2	0	6	6	9	9	11	11	14	14	18
4	6	10	10	19	19	22	22	29	29	32
3	10	11	19	23	23	26	29	37	37	38
1	11	13	23	24	26	34	37	39	39	41
5	13	19	24	32	34	41	41	44	44	47

didapatkan *makespan* (Ms) sebesar 47 dengan total *flowtime* (F) sebesar 176.
Dihtung besar e1(sesuai rumus 2.1), e2 (sesuai rumus 2.2) dan Δ (sesuai rumus 2.3) untuk jadwal 2.1 sebagai berikut :

Ms = 43; Ms new = 47; F = 160; F new = 176; maka

$$e1 = 0{,}5\left(\frac{(43-\min(43{,}47))}{\min(43{,}47)}\right) + 0{,}5\left(\frac{(160-\min(160{,}176))}{\min(160{,}176)}\right)$$

$$= 0$$

$$e2 = 0{,}5\left(\frac{(47-\min(43{,}47))}{\min(43{,}47)}\right) + 0{,}5\left(\frac{(176-\min(160{,}176))}{\min(160{,}176)}\right)$$

$$= 0{,}0465 + 0{,}05 = 0{,}0965$$

$\Delta = e2 - e1 = 0{,}0965 - 0 = 0{,}0965$
Karena $\Delta \geq 0$ yang menandakan bahwa jadwal baru tidak lebih baik dari jadwal lama, maka dihitung *probabilitas* (sesuai rumus 2.4) penerimaan sebagai berikut :

$$p = \exp\left(-\frac{0{,}0965}{0{,}35}\right) = \exp(-0{,}2757) = 0{,}759$$

diambil angka random antara 0 hingga 1, misal didapat px = 0.435. Karena px ≤ p, maka jadwal baru tersebut bisa diterima.

- **Iterasi 2.2**

  T_0 = 0,35; R = 2; Sa = 2-4-3-1-5;

  Makespan (Ms) = 47; total *flowtime* (F) = 176

  Mencari 2 buah nilai random r1 dan r2 antara 1 hingga 5 (sejumlah *job*). Misal didapatkan r1 = 2 dan r2 = 4, maka tukarkan posisi *job* ke-2 dan *job* ke-4.

  Sb : 2-1-3-4-5

  Penjadwalan iterasi 2.2

Tabel 2.12 Penjadwalan Iterasi 2.2 (per satuan waktu)

Job	Mesin 1		Mesin 2		Mesin 3		Mesin 4		Mesin 5	
	S	E	S	E	S	E	S	E	S	E
2	0	6	6	9	9	11	11	14	14	18
1	6	8	9	10	11	19	19	21	21	23
3	8	9	10	14	19	22	22	30	30	31
4	9	13	14	23	23	26	30	37	37	40
5	13	19	23	31	31	38	38	41	41	44

didapatkan *makespan* (Ms) sebesar 44 dengan total *flowtime* (F) sebesar 156.

Dihtung besar e1 (sesuai rumus 2.1), e2 (sesuai rumus 2.2) dan Δ (sesuai rumus 2.3) untuk jadwal 2.2 sebagai berikut :

Ms = 47; Ms new = 44; F = 176; F new = 156; maka

$$e1 = 0{,}5\ \left(\frac{(47-\min(47{,}44))}{\min\ (47{,}44)}\right) + 0{,}5\ \left(\frac{(176-\min(176{,}156))}{\min\ (176{,}156)}\right)$$

$$= 0{,}034 + 0{,}0641 = 0{,}0981$$

$$e2 = 0{,}5\ \left(\frac{(44-\min(47{,}44))}{\min\ (47{,}44)}\right) + 0{,}5\ \left(\frac{(156-\min(176{,}156))}{\min\ (176{,}156)}\right)$$

$$= 0$$

Δ = e2 – e1 = 0 – 0,0981 = - 0,0981

Karena Δ ≤ 0, maka jadwal baru langsung diterima.

Pada parameter awal telah ditentukan nilai *replikasi* sebesar 2 dan saat ini nilai R = 2, maka suhu harus diturunkan sesuai dengan faktor reduksi (sesuai rumus 2.5) yang telah ditentukan.

- **Iterasi 3.1**

  $T_0 = 0{,}35 \times 0{,}7 = 0{,}245$; R = 1; Sa = 2-1-3-4-5;

  Makespan (Ms) = 44; total *flowtime* (F) = 156

  Mencari 2 buah nilai random r1 dan r2 antara 1 hingga 5 (sejumlah *job*). Misal didapatkan r1 = 5 dan r2 = 2, maka tukarkan posisi *job* ke-5 dan *job* ke-2.

  Sb : 2-5-3-4-1

  Penjadwalan iterasi 3.1

Tabel 2.13 Penjadwalan Iterasi 3.1 (per satuan waktu)

Job	Mesin 1		Mesin 2		Mesin 3		Mesin 4		Mesin 5	
	S	E	S	E	S	E	S	E	S	E
2	0	6	6	9	9	11	11	14	14	18
5	6	12	12	20	20	27	27	30	30	33
3	12	13	20	24	27	30	30	38	38	39
4	13	17	24	33	33	36	38	45	45	48
1	17	19	33	34	36	44	45	47	48	50

didapatkan *makespan* (Ms) sebesar 50 dengan total *flowtime* (F) sebesar 188.

Dihtung besar e1 (sesuai rumus 2.1), e2 (sesuai rumus 2.2) dan Δ (sesuai rumus 2.3) untuk jadwal 3.1 sebagai berikut :

Ms = 44; Ms new = 50; F = 156; F new = 188; maka

$$e1 = 0{,}5\left(\frac{(44-\min(44{,}50))}{\min(44{,}50)}\right) + 0{,}5\left(\frac{(156-\min(156{,}188))}{\min(156{,}188)}\right)$$

$= 0$

$$e2 = 0{,}5\left(\frac{(50-\min(44{,}50))}{\min(44{,}50)}\right) + 0{,}5\left(\frac{(188-\min(156{,}188))}{\min(156{,}188)}\right)$$

$= 0{,}0682 + 0{,}1025 = 0{,}1707$

$\Delta = e2 - e1 = 0{,}1707 - 0 = 0{,}1707$

Karena $\Delta \geq 0$ yang menandakan bahwa jadwal baru tidak lebih baik dari jadwal lama, maka dihitung *probabilitas* (sesuai rumus 2.4) penerimaan sebagai berikut :

$$p = \exp(-\frac{0,1707}{0,245}) = \exp(-0,6967) = 0,4982$$

diambil angka random antara 0 hingga 1, misal didapat px = 0.823. Karena px $\geq$ p, maka jadwal baru tersebut ditolak.

- **Iterasi 3.2**

T_0 = 0,245; R = 2; Sa = 2-1-3-4-5;
Makespan (Ms) = 44; total *flowtime* (F) = 156
Mencari 2 buah nilai random r1 dan r2 antara 1 hingga 5 (sejumlah *job*). Misal didapatkan r1 = 1 dan r2 = 3, maka tukarkan posisi *job* ke-1 dan *job* ke-3.
Sb : 3-1-2-4-5
Penjadwalan iterasi 3.2

Tabel 2.14 Penjadwalan Iterasi 3.2 (per satuan waktu)

Job	Mesin 1		Mesin 2		Mesin 3		Mesin 4		Mesin 5	
	S	E	S	E	S	E	S	E	S	E
3	0	1	1	5	5	8	8	16	16	17
1	1	3	5	6	8	16	16	18	18	20
2	3	9	9	12	16	18	18	21	21	25
4	9	13	13	22	22	25	25	32	32	35
5	13	19	22	30	30	37	37	40	40	43

didapatkan *makespan* (Ms) sebesar 43 dengan total *flowtime* (F) sebesar 140.
Dihtung besar e1 (sesuai rumus 2.1), e2 (sesuai rumus 2.2) dan Δ (sesuai rumus 2.3) untuk jadwal 3.2 sebagai berikut :

Ms = 44; Ms new = 43; F = 156; F new = 140; maka

$$e1 = 0,5 \left(\frac{(44-\min(44,43))}{\min(44,43)}\right) + 0,5 \left(\frac{(156-\min(156,140))}{\min(156,140)}\right)$$

$$= 0,0116 + 0,0571 = 0,0687$$

$$e2 = 0{,}5\left(\frac{(43-\min(44{,}43))}{\min(44{,}43)}\right) + 0{,}5\left(\frac{(140-\min(156{,}140))}{\min(156{,}140)}\right)$$

$= 0$

$\Delta = e2 - e1 = 0 - 0{,}0687 = -0{,}0687$

Karena $\Delta \leq 0$, maka jadwal baru langsung diterima.

Pada parameter awal telah ditentukan nilai *replikasi* sebesar 2 dan saat ini nilai R = 2, maka suhu harus diturunkan sesuai dengan faktor reduksi (sesuai rumus 2.5) yang telah ditentukan.

- **Iterasi 4.1**

  $T_0 = 0{,}245 \times 0{,}7 = 0{,}1715$; R = 1

  Karena T_0 sudah berada di bawah Ta yang sebesar 0,23, maka ***iterasi dihentikan***. Jadi untuk masalah penjadwalan di atas menghasilkan solusi akhir jadwal 3-1-2-4-5 dengan *makespan* (Ms) sebesar 43 dan total *flowtime* (F) sebesar 140.

2.6 Algoritma Genetika

2.6.1. Pengertian

Algoritma genetika adalah algoritma pencarian solusi *neighborhood* berdasarkan mekanisme seleksi alam (*natural selection*) dan genetika alam (*natural genetics*) yang dapat digunakan untuk memecahkan *combinatorial optimization problems* yang sulit. Algoritma genetika dikembangkan oleh John Holland (1975) yang mengkombinasikan keberhasilan suatu struktur untuk bertahan hidup (*survival of fittest*) dengan pengubahan informasi dari struktur tersebut secara random untuk membentuk suatu mekanisme penelusuran seperti yang dilakukan dalam proses pembentukan manusia atau makhluk hidup (*gen* / sifat yang diturunkan).[Medianti,1999].

Dalam algoritma genetika dikenal adanya *chromosome* yaitu kandidat penyelesaian yang digambarkan dengan urutan *binary digits* atau *integers* sesuai kondisi yang dikehendaki. *Chromosome* terdiri dari *gen-gen* yang melambangkan ciri-ciri unik dari *chromosome* tersebut, sedangkan nilai yang berkaitan dengan ciri-ciri yang dilambangkan oleh *gen* tertentu disebut *allele*. Populasi adalah sekumpulan *chromosome* yang akan diperbarui pada setiap generasi. [Medianti,1999].

Dikenal pula adanya dua operator yang memegang peranan penting dalam algoritma genetika yaitu *crossover operator* dan *mutation operator*. *Crossover operator* adalah operator yang menggabungkan dua *chromosome* sehingga menghasilkan *child chromosome* yang mewarisi ciri-ciri dasar dari *parent chromosome*. Sedangkan *mutation operator* digunakan untuk memperkenalkan transformasi (informasi) baru pada populasi yang melibatkan perubahan *chromosome* secara random. Jika algoritma genetika diaplikasikan pada problem penjadwalan maka *chromosome* menggambarkan urutan *job*, *gen-gen* melambangkan posisi-posisi *job* tersebut, sedangkan *allele* menggambarkan *job-job* yang memiliki posisi-posisi tersebut.[Rajendran,1992]

Algoritma genetika sangat berbeda dengan algoritma optimasi dan pencarian yang lainnya pada beberapa hal, diantaranya : [Goldberg,1989]

1. Algoritma genetika bekerja dalam bentuk kode dengan sekumpulan parameter, bukan parameter itu sendiri.
2. Algoritma genetika mencari solusi dari sekumpulan populasi, bukan dari satu titik solusi.
3. Algoritma genetika langsung menggunakan informasi fungsi objektif (berupa fungsi tujuan) dan bukan turunannya.
4. Algoritma genetika menggunakan aturan *probabilitas*, bukan aturan deterministik.

2.6.2. Istilah dan Operator Genetika

Terdapat beberapa istilah dan operator yang umumnya digunakan dalam algoritma genetika, yaitu : [Hermawanto, 2003]

1) *Chromosome*

Sebuah solusi yang dibangkitkan dalam algoritma genetika. Sedangkan kumpulan dari *chromosome-chromosome* itu disebut populasi.

2) *Generasi*

Sebuah *chromosome* dibentuk dari komponen-komponen penyusun yang disebut sebagai *gen* dan nilainya dapat berupa bilangan numerik, biner, simbol ataupun karakter tergantung dari permasalahan yang ingin diselesaikan. *Chromosome-chromosome* tersebut akan ber-evolusi secara berkelanjutan yang disebut generasi.

3) *Fitness*

Dalam tiap generasi, *chromosome-chromosome* tersebut dievaluasi tingkat keberhasilan nilai solusinya terhadap masalah yang ingin diselesaikan (nilai objektif) menggunakan ukuran yang disebut dengan *fitness*.

4) *Seleksi*

Untuk memilih *chromosome* yang tetap dipertahankan untuk *generasi* selanjutnya dilakukan proses yang disebut dengan seleksi. Proses seleksi *chromosome* menggunakan konsep aturan evolusi Darwin yang telah disebutkan sebelumnya yaitu *chromosome* yang mempunyai nilai *fitness* tinggi akan memiliki peluang lebih besar untuk terpilih lagi pada generasi selanjutnya.

5) *Crossover*

Crossover merupakan bagian terpenting dalam algoritma genetika, karena disini ditentukan bagaimana membentuk generasi yang baru. *Crossover* adalah mekanisme diversifikasi yang dimiliki algoritma genetika, yang merupakan operator yang menggabungkan dua *chromosome* sehingga menghasilkan *child chromosome* yang mewarisi ciri-ciri dasar dari *parent chromosome.*

6) *Mutasi*

Mutasi digunakan karena walau *reproduksi* dan *crossover* dapat secara efektif melakukan semua pencarian yang diharapkan. Tujuan lain adalah untuk tetap bertahan agar tidak terjebak dalam optimum lokal dan untuk memperkenalkan transformasi (informasi) baru pada populasi yang melibatkan perubahan *chromosome* secara random.

2.6.3. Skema

Skema umum dalam algoritma genetika yaitu : [Dawei, 1999]

1. Menginisialisasi populasi awal P_0 (biasanya didapatkan dari algoritma heuristik yang lebih sederhana ataupun dengan random)
2. Mengevaluasi populasi P_0
3. Menginisialisasi generasi = 1
4. Mengerjakan langkah-langkah berikut sampai kondisi tertentu :
    - *Crossover* $P_{generasi}$
    - *Mutasi* $P_{generasi}$

- *Evaluasi* P $_{generasi}$
- *Generasi* = *generasi* + 1

2.6.4. Contoh Manual

Permasalahan penjadwalan yang sama seperti algoritma *simulated annealing* akan diselesaikan dengan algoritma genetika.

- **Parameter Awal**
  - Jumlah individu dalam 1 *generasi* = 4
  - Jumlah *generasi* = 4
  - Probabilitas *mutasi* = 0,1

- **Iterasi 1**

  Hasil dari algoritma *Shortest Processing Time* (SPT) yaitu 1-3-2-4-5 dengan *makespan* (Ms) = 43 dan total *flowtime* (F) = 145. Hasil ini menjadi *initial solution* bagi algoritma genetika.
  P1 : 1-3-2-4-5; *makespan* (Ms) = 43 dan total *flowtime* (F) = 145.

  Dengan cara random didapatkan *parent-parent* berikutnya sebagai berikut :

  P2 : 2-3-4-1-5

  Penjadwalan P2

Tabel 2.15 Penjadwalan P2 (per satuan waktu)

Job	Mesin 1		Mesin 2		Mesin 3		Mesin 4		Mesin 5	
	S	E	S	E	S	E	S	E	S	E
2	0	6	6	9	9	11	11	14	14	18
3	6	7	9	13	13	16	16	24	24	25
4	7	11	13	22	22	25	25	32	32	35
1	11	13	22	23	25	33	33	35	35	37
5	13	19	23	31	33	40	40	43	43	46

Untuk P2 didapat *makespan*(Ms) = 46 dan total *flowtime*(F) = 161

P3 : 3-1-5-2-4

Penjadwalan P3

Tabel 2.16 Penjadwalan P3 (per satuan waktu)

Job	Mesin 1		Mesin 2		Mesin 3		Mesin 4		Mesin 5	
	S	E	S	E	S	E	S	E	S	E
3	0	1	1	5	5	8	8	16	16	17
1	1	3	5	6	8	16	16	18	18	20
5	3	9	9	17	17	24	24	27	27	30
2	9	15	17	20	24	26	27	30	30	34
4	15	19	20	29	29	32	32	39	39	42

Untuk P3 didapat *makespan*(Ms) = 42 dan total *flowtime*(F) = 143

P4 : 1-4-2-5-3

Penjadwalan P4

Tabel 2.17 Penjadwalan P4 (per satuan waktu)

Job	Mesin 1		Mesin 2		Mesin 3		Mesin 4		Mesin 5	
	S	E	S	E	S	E	S	E	S	E
1	0	2	2	3	3	11	11	13	13	15
4	2	6	6	15	15	18	18	25	25	28
2	6	12	15	18	18	20	25	28	28	32
5	12	18	18	26	26	33	33	36	36	39
3	18	19	26	30	33	36	36	44	44	45

Untuk P4 didapat *makespan*(Ms) = 45 dan total *flowtime*(F) = 159

Generasi 1 :

P1 : 1-3-2-4-5; *makespan* (Ms) = 43; total *flowtime* (F) = 145
P2 : 2-3-4-1-5; *makespan* (Ms) = 46; total *flowtime* (F) = 161
P3 : 3-1-5-2-4; *makespan* (Ms) = 42; total *flowtime* (F) = 143
P4 : 1-4-2-5-3; *makespan* (Ms) = 45; total *flowtime* (F) = 159

Untuk generasi 1 tidak perlu dihitung nilai objektif keseluruhannya (Y), karena generasi 1 sudah berjumlah 4 (sejumlah *parent*) dan tidak perlu diseleksi mana yang terbaik karena semua akan mengalami *crossover*.

- ✓ **P1 crossover P2**

Mengambil 2 angka random antara 1 hingga 5. Misal r1 = 2 dan r2 = 4.

P1 : 1-|3-2-4|-5 → Pa : 1-|3-4-1|-5
P2 : 2-|3-4-1|-5 → Pb : 2-|3-2-4|-5

Hasil crossover antara P1 dan P2 di atas menghasilkan Pa dan Pb, dimana Pa : 1-3-4-1-5 dan Pb : 2-3-2-4-5. Dari hasil tersebut dapat dilihat bahwa masih ada gen yang sama dalam 1 kromosom, maka langkah yang harus dilakukan yaitu melakukan repair kromosom. Untuk kromosom Pa, hasil repair kromosom adalah 1-3-4-2-5 dan hasil repair kromosom Pb adalah 2-3-1-4-5.

Pa : 1-|3-4-1|-5 → Pa : 1-3-4-2-5
Pb : 2-|3-2-4|-5 → Pb : 2-3-1-4-5

Penjadwalan Pa

Tabel 2.18 Penjadwalan Pa (per satuan waktu)

Job	Mesin 1		Mesin 2		Mesin 3		Mesin 4		Mesin 5	
	S	E	S	E	S	E	S	E	S	E
1	0	2	2	3	3	11	11	13	13	15
3	2	3	3	7	11	14	14	22	22	23
4	3	7	7	16	16	19	22	29	29	32
2	7	13	16	19	19	21	29	32	32	36
5	13	19	19	27	27	34	34	37	37	40

Untuk Pa didapatkan *makespan* (Ms) = 40 dan total *flowtime* (F) = 146

Penjadwalan Pb

Tabel 2.19 Penjadwalan Pb (per satuan waktu)

Job	Mesin 1		Mesin 2		Mesin 3		Mesin 4		Mesin 5	
	S	E	S	E	S	E	S	E	S	E
2	0	6	6	9	9	11	11	14	14	18
3	6	7	9	13	13	16	16	24	24	25
1	7	9	13	14	16	24	24	26	26	28
4	9	13	14	23	24	27	27	34	34	37
5	13	19	23	31	31	38	38	41	41	44

Untuk Pb didapatkan *makespan* (Ms) = 44 dan total *flowtime* (F) = 152

✓ **P1 crossover P3**

Mengambil 2 angka random, misal r1 = 3 dan r2 = 5.

P1 : 1-3-|2-4-5| → Pc : 1-3-|5-2-4|
P3 : 3-1-|5-2-4| → Pd : 3-1-|2-4-5|

Hasil crossover antara P1 dan P3 di atas menghasilkan Pc dan Pd, dimana Pc : 1-3-5-2-4 dan Pd : 3-1-2-4-5. Dari hasil tersebut dapat dilihat bahwa tidak ada gen yang sama dalam 1 kromosom, maka tidak dilakukan repair kromosom.

Pc : 1-3-|5-2-4| → Pc : 1-3-5-2-4
Pd : 3-1-|2-4-5| → Pd : 3-1-2-4-5

Penjadwalan Pc

Tabel 2.20 Penjadwalan Pc (per satuan waktu)

Job	Mesin 1		Mesin 2		Mesin 3		Mesin 4		Mesin 5	
	S	E	S	E	S	E	S	E	S	E
1	0	2	2	3	3	11	11	13	13	15
3	2	3	3	7	11	14	14	22	22	23
5	3	9	9	17	17	24	24	27	27	30
2	9	15	17	20	24	26	27	30	30	34
4	15	19	20	29	29	32	32	39	39	42

Untuk Pc didapatkan *makespan* (Ms) = 42 dan total *flowtime* (F) = 144

Penjadwalan Pd

Tabel 2.21 Penjadwalan Pd (per satuan waktu)

Job	Mesin 1		Mesin 2		Mesin 3		Mesin 4		Mesin 5	
	S	E	S	E	S	E	S	E	S	E
3	0	1	1	5	5	8	8	16	16	17
1	1	3	5	6	8	16	16	18	18	20
2	3	9	9	12	16	18	18	21	21	25
4	9	13	13	22	22	25	25	32	32	35
5	13	19	22	30	30	37	37	40	40	43

Untuk Pd didapatkan *makespan* (Ms) = 43 dan total *flowtime* (F) = 140

✓ **P1 crossover P4**

Mengambil 2 angka random, misal r1 = 1 dan r2 = 4.

P1 : |1-3-2-4|-5 → Pe : |1-4-2-5|-5
P4 : |1-4-2-5|-3 → Pf : |1-3-2-4|-3

Hasil crossover antara P1 dan P4 di atas menghasilkan Pe dan Pf, dimana Pe : 1-4-2-5-5 dan Pf : 1-3-2-4-3. Dari hasil tersebut dapat dilihat bahwa masih ada gen yang sama dalam 1 kromosom, maka langkah yang harus dilakukan yaitu melakukan repair kromosom. Untuk kromosom Pe, hasil repair kromosom adalah 1-4-2-3-5 dan hasil repair kromosom Pf adalah 1-5-2-4-3.

Pe : |1-4-2-5|-5 → Pe : 1-4-2-3-5
Pf : |1-3-2-4|-3 → Pf : 1-5-2-4-3

Penjadwalan Pe

Tabel 2.22 Penjadwalan Pe (per satuan waktu)

Job	Mesin 1		Mesin 2		Mesin 3		Mesin 4		Mesin 5	
	S	E	S	E	S	E	S	E	S	E
1	0	2	2	3	3	11	11	13	13	15
4	2	6	6	15	15	18	18	25	25	28
2	6	12	15	18	18	20	25	28	28	32
3	12	13	18	22	22	25	28	36	36	37
5	13	19	22	30	30	37	37	40	40	43

Untuk Pe didapatkan *makespan* (Ms) = 43 dan total *flowtime* (F) = 155

Penjadwalan Pf

Tabel 2.23 Penjadwalan Pf (per satuan waktu)

Job	Mesin 1		Mesin 2		Mesin 3		Mesin 4		Mesin 5	
	S	E	S	E	S	E	S	E	S	E
1	0	2	2	3	3	11	11	13	13	15
5	2	8	8	16	16	23	23	26	26	29
2	8	14	16	19	23	25	26	29	29	33
4	14	18	19	28	28	31	31	38	38	41
3	18	19	28	32	32	35	38	46	46	47

Untuk Pf didapatkan *makespan* (Ms) = 47 dan total *flowtime* (F) = 165

- ✓ **P2 crossover P3**

Mengambil 2 angka random, misal r1 = 3 dan r2 = 5.

P2 : 2-3-|4-1-5| → Pg : 2-3-|5-2-4|
P3 : 3-1-|5-2-4| → Ph : 3-1-|4-1-5|

Hasil crossover antara P2 dan P3 di atas menghasilkan Pg dan Ph, dimana Pg : 2-3-5-2-4 dan Ph : 3-1-4-1-5. Dari hasil tersebut dapat dilihat bahwa masih ada gen yang sama dalam 1 kromosom, maka langkah yang harus dilakukan yaitu melakukan repair kromosom. Untuk kromosom Pg, hasil repair kromosom adalah 2-3-5-1-4 dan hasil repair kromosom Ph adalah 3-1-4-2-5.

Pg : 2-3-|5-2-4| → Pg : 2-3-5-1-4
Ph : 3-1-|4-1-5| → Ph : 3-1-4-2-5

Penjadwalan Pg

Tabel 2.24 Penjadwalan Pg (per satuan waktu)

Job	Mesin 1		Mesin 2		Mesin 3		Mesin 4		Mesin 5	
	S	E	S	E	S	E	S	E	S	E
2	0	6	6	9	9	11	11	14	14	18
3	6	7	9	13	13	16	16	24	24	25
5	7	13	13	21	21	28	28	31	31	34
1	13	15	21	22	28	36	36	38	38	40
4	15	19	22	31	36	39	39	46	46	49

Untuk Pg didapatkan *makespan* (Ms) = 49 dan total *flowtime* (F) = 166

Penjadwalan Ph

Tabel 2.25 Penjadwalan Ph (per satuan waktu)

Job	Mesin 1		Mesin 2		Mesin 3		Mesin 4		Mesin 5	
	S	E	S	E	S	E	S	E	S	E
3	0	1	1	5	5	8	8	16	16	17
1	1	3	5	6	8	16	16	18	18	20
4	3	7	7	16	16	19	19	26	26	29
2	7	13	16	19	19	21	26	29	29	33
5	13	19	19	27	27	34	34	37	37	40

Untuk Ph didapatkan *makespan* (Ms) = 40 dan total *flowtime* (F) = 139

✓ **P2 crossover P4**

Mengambil 2 angka random, misal r1 = 5 dan r2 = 2.

P2 : 2-|3-4-1-5| → Pi : 2-|4-2-5-3|
P4 : 1-|4-2-5-3| → Pj : 1-|3-4-1-5|

Hasil crossover antara P2 dan P4 di atas menghasilkan Pi dan Pj, dimana Pi : 2-4-2-5-3 dan Pj : 1-3-4-1-5. Dari hasil tersebut dapat dilihat bahwa masih ada gen yang sama dalam 1 kromosom, maka langkah yang harus dilakukan yaitu melakukan repair kromosom. Untuk kromosom Pi, hasil repair kromosom adalah 2-4-1-5-3 dan hasil repair kromosom Pj adalah 1-3-4-2-5.

Pi : 2-|4-2-5-3| → Pi : 2-4-1-5-3
Pj : 1-|3-4-1-5| → Pj : 1-3-4-2-5

Penjadwalan Pi

Tabel 2.26 Penjadwalan Pi (per satuan waktu)

Job	Mesin 1		Mesin 2		Mesin 3		Mesin 4		Mesin 5	
	S	E	S	E	S	E	S	E	S	E
2	0	6	6	9	9	11	11	14	14	18
4	6	10	10	19	19	22	22	29	29	32
1	10	12	19	20	22	30	30	32	32	34
5	12	18	20	28	30	37	37	40	40	43
3	18	19	28	32	37	40	40	48	48	49

Untuk Pi didapatkan *makespan* (Ms) = 49 dan total *flowtime* (F) = 176

Penjadwalan Pj

Tabel 2.27 Penjadwalan Pj (per satuan waktu)

Job	Mesin 1		Mesin 2		Mesin 3		Mesin 4		Mesin 5	
	S	E	S	E	S	E	S	E	S	E
1	0	2	2	3	3	11	11	13	13	15
3	2	3	3	7	11	14	14	22	22	23
4	3	7	7	16	16	19	22	29	29	32
2	7	13	16	19	19	21	29	32	32	36
5	13	19	19	27	27	34	34	37	37	40

Untuk Pj didapatkan *makespan* (Ms) = 40 dan total *flowtime* (F) = 146

✓ **P3 crossover P4**

Mengambil 2 angka random, misal r1 = 4 dan r2 = 3.

P3 : 3-1-|5-2|-4 → Pk : 3-1-|2-5|-4
P4 : 1-4-|2-5|-3 → Pl : 1-4-|5-2|-3

Hasil crossover antara P3 dan P4 di atas menghasilkan Pk dan Pl, dimana Pk : 3-1-2-5-4 dan Pl : 1-4-5-2-3. Dari hasil tersebut dapat dilihat bahwa tidak ada gen yang sama dalam 1 kromosom, maka tidak dilakukan repair kromosom.

Pk : 3-1-|2-5|-4 → Pk : 3-1-2-5-4
Pl : 1-4-|5-2|-3 → Pl : 1-4-5-2-3

Penjadwalan Pk

Tabel 2.28 Penjadwalan Pk (per satuan waktu)

Job	Mesin 1		Mesin 2		Mesin 3		Mesin 4		Mesin 5	
	S	E	S	E	S	E	S	E	S	E
3	0	1	1	5	5	8	8	16	16	17
1	1	3	5	6	8	16	16	18	18	20
2	3	9	9	12	16	18	18	21	21	25
5	9	15	15	23	23	30	30	33	33	36
4	15	19	23	32	32	35	35	42	42	45

Untuk Pk didapatkan *makespan* (Ms) = 45 dan total *flowtime* (F) = 143

Penjadwalan Pl

Tabel 2.29 Penjadwalan Pl (per satuan waktu)

Job	Mesin 1		Mesin 2		Mesin 3		Mesin 4		Mesin 5	
	S	E	S	E	S	E	S	E	S	E
1	0	2	2	3	3	11	11	13	13	15
4	2	6	6	15	15	18	18	25	25	28
5	6	12	15	23	23	30	30	33	33	36
2	12	18	23	26	30	32	33	36	36	40
3	18	19	26	30	32	35	36	44	44	45

Untuk Pl didapatkan *makespan* (Ms) = 45 dan total *flowtime* (F) = 164

Tabel 2.30 Hasil Crossover Iterasi 1

Individu	Jadwal	Ms	F
Pa	1-3-4-2-5	40	146
Pb	2-3-1-4-5	44	152
Pc	1-3-5-2-4	42	144
Pd	3-1-2-4-5	43	140
Pe	1-4-2-3-5	43	155
Pf	1-5-2-4-3	47	165
Pg	2-3-5-1-4	49	166
Ph	3-1-4-2-5	40	139
Pi	2-4-1-5-3	49	176
Pj	1-3-4-2-5	40	146
Pk	3-1-2-5-4	45	143
Pl	1-4-5-2-3	45	164

Untuk menentukan generasi 2, akan dicari nilai objektif gabungan dari masing-masing jadwal (Y).

$\sum Ms = 527 \rightarrow Msr = \frac{\sum Ms}{\sum anak} = \frac{527}{12} = 43{,}916$

$\sum F = 1836 \rightarrow Fr = \frac{\sum F}{\sum anak} = \frac{1836}{12} = 153$

Keterangan :

Ms : nilai *makespan*
Msr : nilai *makespan* rata-rata
F : nilai total *flowtime*
Fr : nilai total *flowtime* rata-rata

Menghitung nilai *w1* dan *w2* dapat dilakukan dengan rumus berikut :

$$w1 = \frac{Msr}{Fr} \quad (2.6)$$

$$w2 = 1 - w1 \quad (2.7)$$

maka akan dihasilkan nilai w1 dan w2 seperti di bawah ini :

$$w1 = \frac{Msr}{Fr} = \frac{43{,}916}{153} = 0{,}287 \qquad w2 = 1 - 0{,}287 = 0{,}713$$

dengan rumus $Y = w2\ Ms + w1\ F$ (2.8)

maka dapat dihitung nilai Y (nilai objektif) untuk masing-masing jadwal

Tabel 2.31 Nilai Objektif untuk Iterasi 1

Individu	Jadwal	Ms	F	Y
Pa	1-3-4-2-5	40	146	70,422
Pb	2-3-1-4-5	44	152	74,996
Pc	1-3-5-2-4	42	144	71,274
Pd	3-1-2-4-5	43	140	70,839
Pe	1-4-2-3-5	43	155	75,144

Pf	1-5-2-4-3	47	165	80,866
Pg	2-3-5-1-4	49	166	82,579
Ph	3-1-4-2-5	40	139	68,413
Pi	2-4-1-5-3	49	176	85,449
Pj	1-3-4-2-5	40	146	70,422
Pk	3-1-2-5-4	45	143	73,126
Pl	1-4-5-2-3	45	164	79,153

Kemudian dari hasil nilai objektif di atas diurutkan dari nilai Y yang terkecil hingga nilai Y yang terbesar dan diambil 4 individu yang terbaik, karena sudah ditentukan pada parameter awal bahwa digunakan 4 individu pada tiap generasi. Hasilnya adalah Ph(68,413), Pa(70,422), Pj(70,422), dan Pd(70,839).

Langkah selanjutnya adalah memberlakukan operator *mutasi* pada 4 individu seperti berikut ini :

Probabilitas mutasi untuk Ph = 0,347. Tidak lebih kecil dari *probabilitas mutasi awal* (0,1) maka tidak dimutasi.

Probabilitas mutasi untuk Pa = 0,592. Tidak lebih kecil dari *probabilitas mutasi awal* (0,1) maka tidak dimutasi.

Probabilitas mutasi untuk Pj = 0,618. Tidak lebih kecil dari *probabilitas mutasi awal* (0,1) maka tidak dimutasi.

Probabilitas mutasi untuk Pd = 0,094. Lebih kecil dari *probabilitas mutasi awal* (0,1) maka akan dimutasi.

Pd = 3-1-2-4-5; diambil dua nilai random d1 = 1 dan d2 = 5, maka menukarkan *job* 1 dan *job* 5 menjadi Pd = 3-5-2-4-1

- **Iterasi 2**

Generasi 2 :

P1 : 3-1-4-2-5 *makespan* (Ms) = 40; total *flowtime* (F) = 139
P2 : 1-3-4-2-5 *makespan* (Ms) = 40; total *flowtime* (F) = 146
P3 : 1-3-4-2-5 *makespan* (Ms) = 40; total *flowtime* (F) = 146
P4 : 3-5-2-4-1 *makespan* (Ms) = 42; total *flowtime* (F) = 159

✓ **P1 crossover P2**

Mengambil 2 angka random, misal r1 = 2 dan r2 = 4.

P1 : 3-|1-4-2|-5 → Pa : 3-|3-4-2|-5
P2 : 1-|3-4-2|-5 → Pb : 1-|1-4-2|-5

Hasil crossover antara P1 dan P2 di atas menghasilkan Pa dan Pb, dimana Pa : 3-3-4-2-5 dan Pb : 1-1-4-2-5. Dari hasil tersebut dapat dilihat bahwa masih ada gen yang sama dalam 1 kromosom, maka langkah yang harus dilakukan yaitu melakukan repair kromosom. Untuk kromosom Pa, hasil repair kromosom adalah 3-1-4-2-5 dan hasil repair kromosom Pb adalah 1-3-4-2-5.

Pa : 3-|3-4-2|-5 → Pa : 3-1-4-2-5 Ms = 40 F = 139
Pb : 1-|1-4-2|-5 → Pb : 1-3-4-2-5 Ms = 40 F = 146

✓ **P1 crossover P3**

Mengambil 2 angka random, misal r1 = 1 dan r2 = 3.

P1 : |3-1-4|-2-5 → Pc : |1-3-4|-2-5
P3 : |1-3-4|-2-5 → Pd : |3-1-4|-2-5

Hasil crossover antara P1 dan P3 di atas menghasilkan Pc dan Pd, dimana Pc : 1-3-4-2-5 dan Pd : 3-1-4-2-5. Dari hasil tersebut dapat dilihat bahwa tidak ada gen yang sama dalam 1 kromosom, maka tidak dilakukan repair kromosom.

Pc : |1-3-4|-2-5 → Pc : 1-3-4-2-5 Ms = 40 F = 146
Pd : |3-1-4|-2-5 → Pd : 3-1-4-2-5 Ms = 40 F = 139

✓ **P1 crossover P4**

Mengambil 2 angka random, misal r1 = 3 dan r2 = 5.

P1 : 3-1-|4-2-5| → Pe : 3-1-|2-4-1|
P4 : 3-5-|2-4-1| → Pf : 3-5-|4-2-5|

Hasil crossover antara P1 dan P4 di atas menghasilkan Pe dan Pf, dimana Pe : 3-1-2-4-1 dan Pf : 3-5-4-2-5. Dari hasil tersebut dapat dilihat bahwa masih ada gen yang sama dalam 1 kromosom, maka langkah yang harus dilakukan yaitu melakukan repair kromosom.

Untuk kromosom Pe, hasil repair kromosom adalah 3-1-2-4-5 dan hasil repair kromosom Pf adalah 3-5-4-2-1.

Pe : 3-1-|2-4-1| → Pe : 3-1-2-4-5 Ms = 43 F = 140
Pf : 3-5-|4-2-5| → Pf : 3-5-4-2-1 Ms = 43 F = 166

✓ **P2 crossover P3**

Mengambil 2 angka random, misal r1 = 1 dan r2 = 4.

P2 : |1-3-4-2|-5 → Pg : |1-3-4-2|-5
P3 : |1-3-4-2|-5 → Ph : |1-3-4-2|-5

Hasil crossover antara P2 dan P3 di atas menghasilkan Pg dan Ph, dimana Pg : 1-3-4-2-5 dan Ph : 1-3-4-2-5. Dari hasil tersebut dapat dilihat bahwa tidak ada gen yang sama dalam 1 kromosom, maka tidak dilakukan repair kromosom.

Pg : |1-3-4-2|-5 → Pg : 1-3-4-2-5 Ms = 40 F = 146
Ph : |1-3-4-2|-5 → Ph : 1-3-4-2-5 Ms = 40 F = 146

✓ **P2 crossover P4**

Mengambil 2 angka random, misal r1 = 1 dan r2 = 2.

P2 : |1-3|-4-2-5 → Pi : |3-5|-4-2-5
P4 : |3-5|-2-4-1 → Pj : |1-3|-2-4-1

Hasil crossover antara P2 dan P4 di atas menghasilkan Pi dan Pj, dimana Pi : 3-5-4-2-5 dan Pj : 1-3-2-4-1. Dari hasil tersebut dapat dilihat bahwa masih ada gen yang sama dalam 1 kromosom, maka langkah yang harus dilakukan yaitu melakukan repair kromosom. Untuk kromosom Pi, hasil repair kromosom adalah 3-1-4-2-5 dan hasil repair kromosom Pj adalah 5-3-2-4-1.

Pi : |3-5|-4-2-5 → Pi : 3-1-4-2-5 Ms = 40 F = 139
Pj : |1-3|-2-4-1 → Pj : 5-3-2-4-1 Ms = 47 F = 191

✓ **P3 crossover P4**

Mengambil 2 angka random, misal r1 = 4 dan r2 = 2.

P3 : 1-|3-4-2|-5 → Pk : 1-|5-2-4|-5
P4 : 3-|5-2-4|-1 → Pl : 3-|3-4-2|-1

Hasil crossover antara P3 dan P4 di atas menghasilkan Pk dan Pl, dimana Pk : 1-5-2-4-5 dan Pl : 3-3-4-2-1. Dari hasil tersebut dapat dilihat bahwa masih ada gen yang sama dalam 1 kromosom, maka langkah yang harus dilakukan yaitu melakukan repair kromosom. Untuk kromosom Pk, hasil repair kromosom adalah 1-3-2-4-5 dan hasil repair kromosom Pl adalah 3-5-4-2-1.

Pk : 1-|5-2-4|-5 → Pk : 1-3-2-4-5 Ms = 40 F = 142
Pl : 3-|3-4-2|-1 → Pl : 3-5-4-2-1 Ms = 43 F = 166

Tabel 2.32 Hasil Crossover Iterasi 2

Individu	Jadwal	Ms	F
Pa	3-1-4-2-5	40	139
Pb	1-3-4-2-5	40	146
Pc	1-3-4-2-5	40	146
Pd	3-1-4-2-5	40	139
Pe	3-1-2-4-5	43	140
Pf	3-5-4-2-1	43	166
Pg	1-3-4-2-5	40	146
Ph	1-3-4-2-5	40	146
Pi	3-1-4-2-5	40	139
Pj	5-3-2-4-1	47	191
Pk	1-3-2-4-5	40	142
Pl	3-5-4-2-1	43	166

Untuk menentukan generasi 3, akan dicari nilai objektif gabungan dari masing-masing jadwal (Y).

$\sum Ms = 496 \quad Msr = \frac{496}{12} = 41,333$

$\sum F$ = 1806 Fr = $\frac{1806}{12}$ = 150,5

Keterangan :

Ms : nilai *makespan*
Msr : nilai *makespan* rata-rata
F : nilai total *flowtime*
Fr : nilai total *flowtime* rata-rata

$w1 = \frac{Msr}{Fr} = \frac{41,333}{150,5}$ = 0,2746 w2 = 1 – 0,2746 = 0,7254

dengan rumus Y (sesuai rumus 2.8), maka dapat dihitung nilai Y untuk masing-masing jadwal

Tabel 2.33 Nilai Objektif untuk Iterasi 2

Individu	Jadwal	Ms	F	Y
Pa	3-1-4-2-5	40	139	67,1854
Pb	1-3-4-2-5	40	146	69,1076
Pc	1-3-4-2-5	40	146	69,1076
Pd	3-1-4-2-5	40	139	67,1854
Pe	3-1-2-4-5	43	140	69,6362
Pf	3-5-4-2-1	43	166	76,7758
Pg	1-3-4-2-5	40	146	69,1076
Ph	1-3-4-2-5	40	146	69,1076
Pi	3-1-4-2-5	40	139	67,1854
Pj	5-3-2-4-1	47	191	86,5424
Pk	1-3-2-4-5	40	142	68,0092
Pl	3-5-4-2-1	43	166	76,7758

Kemudian dari hasil nilai objektif di atas diurutkan dari nilai Y yang terkecil hingga nilai Y yang terbesar dan diambil 4 *individu* yang terbaik, karena sudah ditentukan pada parameter awal bahwa digunakan 4 individu pada tiap generasi. Hasilnya adalah Pa(67,1854), Pd(67,1854), Pi(67,1854), dan Pk(68,0092).

Langkah selanjutnya adalah memberlakukan operator *mutasi* pada 4 individu seperti berikut ini :
Probabilitas mutasi untuk Pa = 0,347. Tidak lebih kecil dari *probabilitas mutasi awal* (0,1) maka tidak dimutasi.
Probabilitas mutasi untuk Pd = 0,592. Tidak lebih kecil dari *probabilitas mutasi awal* (0,1) maka tidak dimutasi.
Probabilitas mutasi untuk Pi = 0,618. Tidak lebih kecil dari *probabilitas mutasi awal* (0,1) maka tidak dimutasi.
Probabilitas mutasi untuk Pk = 0,941. Tidak lebih kecil dari *probabilitas mutasi awal* (0,1) maka tidak dimutasi.

- **Iterasi 3**

Generasi 3 :

P1 : 3-1-4-2-5 *makespan* (Ms) = 40; total *flowtime* (F) = 139
P2 : 3-1-4-2-5 *makespan* (Ms) = 40; total *flowtime* (F) = 139
P3 : 3-1-4-2-5 *makespan* (Ms) = 40; total *flowtime* (F) = 139
P4 : 1-3-2-4-5 *makespan* (Ms) = 40; total *flowtime* (F) = 142

✓ **P1 crossover P2**

Angka random berapapun hasil *crossover* akan tetap sama seperti *parent*-nya, maka hasil *crossover*-nya :

Pa : 3-1-4-2-5 *makespan* (Ms) = 40; total *flowtime* (F) = 139
Pb : 3-1-4-2-5 *makespan* (Ms) = 40; total *flowtime* (F) = 139

✓ **P1 crossover P3**

Angka random berapapun hasil *crossover* akan tetap sama seperti *parent*-nya, maka hasil *crossover*-nya :

Pc : 3-1-4-2-5 *makespan* (Ms) = 40; total *flowtime* (F) = 139
Pd : 3-1-4-2-5 *makespan* (Ms) = 40; total *flowtime* (F) = 139

✓ **P1 crossover P4**

Mengambil 2 angka random, misal r1 = 2 dan r2 = 4.

P1 : 3-|1-4-2|-5 → Pe : 3-|3-2-4|-5
P4 : 1-|3-2-4|-5 → Pf : 1-|1-4-2|-5

Hasil crossover antara P1 dan P4 di atas menghasilkan Pe dan Pf, dimana Pe : 3-3-2-4-5 dan Pf : 1-1-4-2-5. Dari hasil tersebut dapat dilihat bahwa masih ada gen yang sama dalam 1 kromosom, maka langkah yang harus dilakukan yaitu melakukan repair kromosom. Untuk kromosom Pe, hasil repair kromosom adalah 3-1-2-4-5 dan hasil repair kromosom Pf adalah 1-3-4-2-5.

Pe : 3-|3-2-4|-5 → Pe : 3-1-2-4-5 Ms = 43 F = 140
Pf : 1-|1-4-2|-5 → Pf : 1-3-4-2-5 Ms = 40 F = 146

- ✓ **P2 crossover P3**

Angka random berapapun hasil *crossover* akan tetap sama seperti *parent*-nya, maka hasil *crossover*-nya :

Pg : 3-1-4-2-5 *makespan* (Ms) = 40; total *flowtime* (F) = 139
Ph : 3-1-4-2-5 *makespan* (Ms) = 40; total *flowtime* (F) = 139

- ✓ **P2 crossover P4**

Mengambil 2 angka random, misal r1 = 1 dan r2 = 3.

P2 : |3-1-4|-2-5 → Pi : |1-3-2|-2-5
P4 : |1-3-2|-4-5 → Pj : |3-1-4|-4-5

Hasil crossover antara P2 dan P4 di atas menghasilkan Pi dan Pj, dimana Pi : 1-3-2-2-5 dan Pj : 3-1-4-4-5. Dari hasil tersebut dapat dilihat bahwa masih ada gen yang sama dalam 1 kromosom, maka langkah yang harus dilakukan yaitu melakukan repair kromosom. Untuk kromosom Pi, hasil repair kromosom adalah 1-3-4-2-5 dan hasil repair kromosom Pj adalah 3-1-2-4-5.

Pi : |1-3-2|-2-5 → Pi : 1-3-4-2-5 Ms = 40 F = 146
Pj : |3-1-4|-4-5 → Pj : 3-1-2-4-5 Ms = 43 F = 140

✓ **P3 crossover P4**

Mengambil 2 angka random, misal r1 = 3 dan r2 = 5.

P3 : 3-1-|4-2-5| → Pk : 3-1-|2-4-5|
P4 : 1-3-|2-4-5| → Pl : 1-3-|4-2-5|

Hasil crossover antara P3 dan P4 di atas menghasilkan Pk dan Pl, dimana Pk : 3-1-2-4-5 dan Pl : 1-3-4-2-5. Dari hasil tersebut dapat dilihat bahwa tidak ada gen yang sama dalam 1 kromosom, maka tidak dilakukan repair kromosom.

Pk : 3-1-|2-4-5| → Pk : 3-1-2-4-5 Ms = 43 F = 140
Pl : 1-3-|4-2-5| → Pl : 1-3-4-2-5 Ms = 40 F = 146

Tabel 2.34 Hasil crossover iterasi 3

Individu	Jadwal	Ms	F
Pa	3-1-4-2-5	40	139
Pb	3-1-4-2-5	40	139
Pc	3-1-4-2-5	40	139
Pd	3-1-4-2-5	40	139
Pe	3-1-2-4-5	43	140
Pf	1-3-4-2-5	40	146
Pg	3-1-4-2-5	40	139
Ph	3-1-4-2-5	40	139
Pi	1-3-4-2-5	40	146
Pj	3-1-2-4-5	43	140
Pk	3-1-2-4-5	43	140
Pl	1-3-4-2-5	40	146

Untuk menentukan generasi 3, akan dicari nilai objektif gabungan dari masing-masing jadwal (Y).

$\sum Ms = 496 \quad Msr = \frac{496}{12} = 41{,}333$

$\sum F = 1806 \quad Fr = \frac{1806}{12} = 150{,}5$

Keterangan :

Ms : nilai *makespan*
Msr : nilai *makespan* rata-rata
F : nilai total *flowtime*
Fr : nilai total *flowtime* rata-rata

$w1 = \frac{Msr}{Fr} = \frac{41{,}333}{150{,}5} = 0{,}2746 \quad w2 = 1 - 0{,}2746 = 0{,}7254$

dengan rumus Y (sesuai rumus 2.8), maka dapat dihitung nilai Y untuk masing-masing jadwal

Tabel 2.35 Nilai Objektif untuk Iterasi 3

Individu	Jadwal	Ms	F	Y
Pa	3-1-4-2-5	40	139	67,1854
Pb	3-1-4-2-5	40	139	67,1854
Pc	3-1-4-2-5	40	139	67,1854
Pd	3-1-4-2-5	40	139	67,1854
Pe	3-1-2-4-5	43	140	69,6362
Pf	1-3-4-2-5	40	146	69,1076
Pg	3-1-4-2-5	40	139	67,1854
Ph	3-1-4-2-5	40	139	67,1854
Pi	1-3-4-2-5	40	146	69,1076
Pj	3-1-2-4-5	43	140	69,6362
Pk	3-1-2-4-5	43	140	69,6362
Pl	1-3-4-2-5	40	146	69,1076

Kemudian dari hasil nilai objektif di atas diurutkan dari nilai Y yang terkecil hingga nilai Y yang terbesar dan diambil 4 individu yang terbaik, karena sudah ditentukan pada parameter awal bahwa digunakan 4 individu pada tiap generasi. Hasilnya adalah Pa(67,1854), Pb(67,1854), Pc(67,1854), dan Pd(67,1854).

Langkah selanjutnya adalah memberlakukan operator *mutasi* pada 4 individu seperti berikut ini :
Probabilitas mutasi untuk Pa = 0,147. Tidak lebih kecil dari *probabilitas mutasi awal* (0,1) maka tidak dimutasi.
Probabilitas mutasi untuk Pb = 0,582. Tidak lebih kecil dari *probabilitas mutasi awal* (0,1) maka tidak dimutasi.
Probabilitas mutasi untuk Pc = 0,611. Tidak lebih kecil dari *probabilitas mutasi awal* (0,1) maka tidak dimutasi.
Probabilitas mutasi untuk Pd = 0,241. Tidak lebih kecil dari *probabilitas mutasi awal* (0,1) maka tidak dimutasi.

- **Iterasi 4**

Generasi 4 :

P1 : 3-1-4-2-5 *makespan* (Ms) = 40; *total flowtime* (F) = 139
P2 : 3-1-4-2-5 *makespan* (Ms) = 40; *total flowtime* (F) = 139
P3 : 3-1-4-2-5 *makespan* (Ms) = 40; *total flowtime* (F) = 139
P4 : 3-1-4-2-5 *makespan* (Ms) = 40; *total flowtime* (F) = 139

Karena semua jadwal adalah sama, maka di-*crossover* bagaimanapun akan menghasilkan jadwal yang sama pula, karena itu operator *crossover* dapat dilalui dan langsung ke operator *mutasi*.

Langkah selanjutnya adalah memberlakukan operator *mutasi* pada 4 individu seperti berikut ini :
Probabilitas mutasi untuk P1 = 0,126. Tidak lebih kecil dari *probabilitas mutasi awal* (0,1) maka tidak dimutasi.
Probabilitas mutasi untuk P2 = 0,354. Tidak lebih kecil dari *probabilitas mutasi awal* (0,1) maka tidak dimutasi.
Probabilitas mutasi untuk P3 = 0,102. Tidak lebih kecil dari *probabilitas mutasi awal* (0,1) maka tidak dimutasi.
Probabilitas mutasi untuk P4 = 0,887. Tidak lebih kecil dari *probabilitas mutasi awal* (0,1) maka tidak dimutasi.

Karena iterasi sudah sama dengan jumlah generasi yang ditetapkan pada parameter awal, maka ***iterasi dihentikan***. Oleh karena itu hasil akhir dari algoritma genetika ini adalah jadwal pertama pada generasi terakhir yaitu P1 : 3-1-4-2-5 dengan nilai *makespan* (Ms) = 40 dan total *flowtime* (F) = 139.

2.7 Perbandingan Genetika dan Simulated Annealing

Dalam membandingkan hasil akhir (kinerja) dari dua algoritma tersebut, maka dapat digunakan rumusan sebagai berikut : [Medianti,1999].

$$e1 = w1\left(\frac{(Ms\ GA - \min(Ms\ GA;\ Ms\ SA))}{\min(Ms\ GA;\ Ms\ SA)}\right) +$$

$$w2\left(\frac{F\ GA - \min(F\ GA;\ F\ SA)}{\min(F\ GA;\ F\ SA)}\right)$$

$$e2 = w1\left(\frac{(Ms\ SA - \min(Ms\ GA;\ Ms\ SA))}{\min(Ms\ GA;\ Ms\ SA)}\right) +$$

$$w2\left(\frac{F\ SA - \min(F\ GA;\ F\ SA)}{\min(F\ GA;\ F\ SA)}\right)$$

$$\Delta = e2 - e1 \qquad (2.9)$$

Keterangan :

w1	: bobot *makespan*
w2	: bobot *flowtime*
Ms GA	: nilai *makespan* pada GA (*Genetic Algorithm*)
Ms SA	: nilai *makespan* pada SA (*Simulated Annealing*)
F GA	: nilai total *flowtime* pada GA (*Genetic Algorithm*)
F SA	: nilai total *flowtime* pada SA (*Simulated Annealing*)
e1	: pembobotan *makespan* dan total *flowtime* pada GA (*Genetic Algorithm*) terhadap *makespan* dan total *flowtime* pada SA (*Simulated Annealing*).
e2	: pembobotan *makespan* dan total *flowtime* pada SA (*Simulated Annealing*) terhadap *makespan* dan total *flowtime* pada GA (*Genetic Algorithm*).

Proses perbandingan hasil akhir ini bertujuan untuk mengetahui kinerja algoritma mana yang lebih baik pada suatu percobaan (simulasi). Dari hasil perhitungan Δ (delta) dapat dilihat jika $\Delta \leq 0$, maka SA (*Simulated Annealing*) lebih baik dari GA (*Genetic*

Algorithm) dan sebaliknya jika Δ > 0, maka GA (*Genetic Algorithm*) lebih baik dari SA (*Simulated Annealing*).

2.8 Perhitungan Prosentase Keunggulan Antar Algoritma (Algoritma Genetika terhadap Simulated Annealing atau Simulated Annealing terhadap Algoritma Genetika)

Untuk melihat prosentase keunggulan dari kedua algoritma, maka dapat dihitung dengan rumus sebagai berikut : [Medianti,1999].

Jika *genetic algorithm* (GA) unggul, maka

$$x = \frac{Y\,GA - Y\,SA}{Y\,GA} \times 100\ \% \qquad (2.10)$$

dan jika *simulated annealing* (SA) unggul, maka

$$x = \frac{Y\,SA - Y\,GA}{Y\,SA} \times 100\ \% \qquad (2.11)$$

Keterangan :

x → prosentase keunggulan (baik GA terhadap SA atau sebaliknya, SA terhadap GA)
Y SA → nilai objektif *Simulated Annealing* yang dihasilkan dari perhitungan (w2 * Ms SA) + (w1 * F SA)
Y GA → nilai objektif *Genetic Algorithm* yang dihasilkan dari perhitungan (w2 * Ms GA) + (w1 * F GA), dimana

w1 : bobot *makespan*
w2 : bobot *flowtime*
Ms GA : nilai *makespan* pada GA (*Genetic Algorithm*)
Ms SA : nilai *makespan* pada SA (*Simulated Annealing*)
F GA : nilai total *flowtime* pada GA (*Genetic Algorithm*)
F SA : nilai total *flowtime* pada SA (*Simulated Annealing*)

BAB III
METODOLOGI DAN PERANCANGAN

Pada bab ini akan dibahas tentang metode dan tahap perancangan dalam pembuatan aplikasi *flow shop scheduling* menggunakan algoritma *simulated annealing* dan algoritma genetika.
Adapun tahapan pembuatannya adalah sebagai berikut:

1. Melakukan studi literatur mengenai algoritma *simulated annealing* dan algoritma genetika dalam permasalahan *flow shop scheduling.*
2. Menganalisa dan merancang perangkat lunak.
3. Implementasi perangkat lunak berdasarkan analisa dan perancangan yang dilakukan.
4. Melakukan simulasi terhadap perangkat lunak
5. Melakukan evaluasi (analisa) hasil yang diperoleh dari uji coba tersebut dengan cara membandingkan antara algoritma *simulated annealing* dan algoritma genetika*.*

3.1 Deskripsi Umum Sistem

Pada penjadwalan *flow shop* (*flow shop scheduling*) ini akan dicoba dibandingkan dua algoritma yang biasa digunakan dalam permasalahan scheduling yaitu algoritma *simulated annealing* dan algoritma genetika. Dari hasil uji coba (simulasi) cukup dilakukan pembandingan pada satu jadwal yaitu jadwal yang terbaik dari masing-masing algoritma. Hal yang nantinya akan diperbandingkan menyangkut perhitungan *makespan*, total *flowtime* dan hasil prosentase keunggulan antar algoritma tersebut.

3.2 Analisa Data

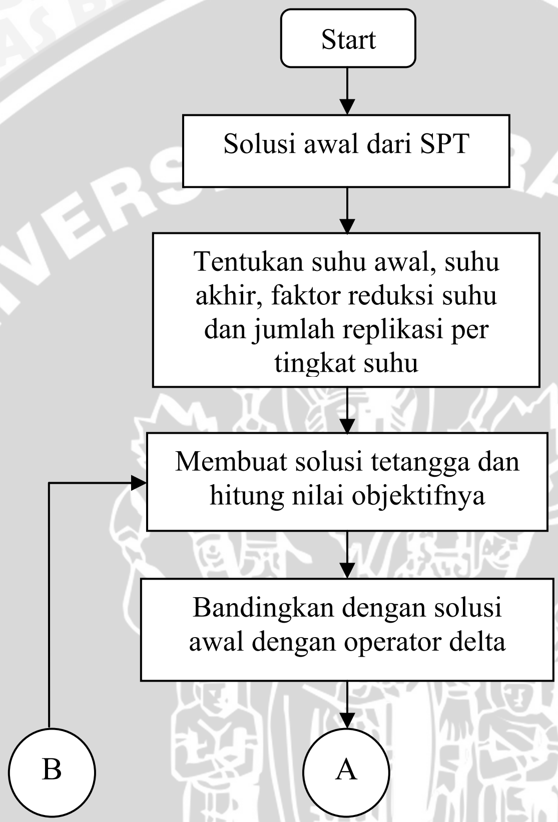
Pada penelitian ini, data yang digunakan dalam pengujian akan bervariasi sesuai input dari user untuk jumlah *job* maupun jumlah mesin yang digunakan. Waktu proses untuk tiap problem dapat bernilai random (otomatis dijalankan komputer) atau user akan menginputkan secara manual.

3.3 Perancangan Proses

Perancangan proses dalam *flow shop scheduling* melalui beberapa tahapan mulai dari perhitungan jadwal menggunakan algoritma *simulated annealing* maupun algoritma genetika sampai perbandingan kinerja dan prosentase keunggulan antar algoritma tersebut

3.3.1. Proses Penjadwalan dengan Simulated Annealing

Beberapa langkah perancangan proses penjadwalan dengan algoritma *simulated annealing* dapat digambarkan dengan flow chart berikut :



Gambar 3.1 Flow Chart Algoritma *Simulated Annealing* [Ponnambalan,1999]

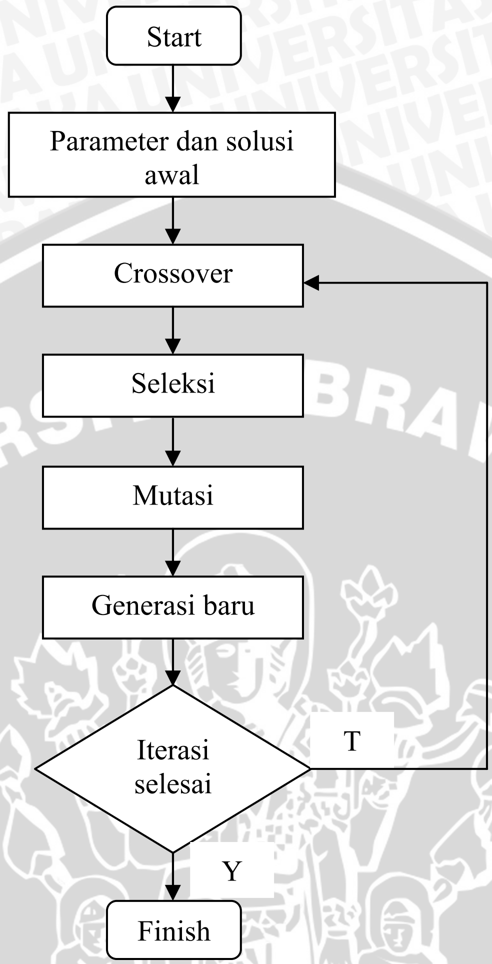
Langkah sistematis dari algoritma *simulated annealing* yaitu

1. Solusi awal didapatkan dari penjadwalan heuristik SPT (*Shortest Processing Time*) dan dihitung nilai objektifnya yaitu *makespan* (*Ms*) dan total *flowtime* (*F*).

2. Menentukan parameter-parameter awal yang dibutuhkan antara lain :
   1) Temperatur awal (T_0), merupakan penanda awal iterasi dimana nantinya temperatur awal ini akan terus berkurang hingga mencapai temperatur akhir.
   2) Temperatur akhir (*Ta*), merupakan batas akhir dari iterasi dimana iterasi sudah dapat dihentikan.
   3) Faktor reduksi suhu (*r*), merupakan angka yang digunakan untuk menurunkan suhu secara bertahap dan terkendali
   4) Angka replikasi (*R*), merupakan angka yang menunjukkan berapa kali loop (perulangan) yang harus dilakukan sebelum menurunkan suhu.
3. Mengambil dua angka random *r1* dan *r2* antara 1 hingga jumlah job, dimana *r1* tidak sama dengan *r2*.
4. Menukarkan *job* pada posisi *r1* dengan *job* pada posisi *r2*.
5. Menghitung nilai objektif jadwal baru, Ms_{new} dan F_{new}
6. Menghitung Δ sesuai dengan rumus 2.3, dimana sebelumnya dicari terlebih dahulu nilai e1 (sesuai rumus 2.1) dan e2 (sesuai rumus 2.2).
7. Jika $\Delta \geq 0$ maka hitung *p* (sesuai rumus 2.4). Ambil satu bilangan random px antara 0 hingga 1, jika px < p maka jadwal baru diterima, jika tidak jadwal lama tetap dipertahankan.
   Jika $\Delta < 0$ maka jadwal baru langsung diterima. Secara umum Δ memiliki pengertian selisih keunggulan jadwal baru terhadap nilai minimum yang dihasilkan dengan jadwal lama terhadap nilai minimum yang dihasilkan.
8. Apabila kondisi replikasi lokal telah dipenuhi, maka lakukan reduksi terhadap suhu (sesuai dengan rumus 2.5)
9. Kembali ke langkah ketiga sampai kriteria penghentian iterasi dipenuhi.

3.3.2. Proses penjadwalan dengan algoritma genetika

Beberapa langkah perancangan proses penjadwalan dengan algoritma genetika dapat digambarkan dengan flow chart berikut :



Gambar 3.2 Flow Chart Algoritma Genetika
[Limyana,2002]

Langkah-langkah spesifik algoritma genetika yang digunakan antara lain :

1. Tentukan semua parameter awal yang dibutuhkan

   Parameter-parameter yang digunakan dalam skema algoritma genetika ini adalah sebagai berikut :

   - Banyak individu dalam 1 generasi; ini dapat ditentukan sendiri tanpa ada aturan tertentu. Namun pada jumlah *job* yang banyak, sebaiknya individu dalam 1 generasi juga banyak agar pencarian solusi akan lebih baik.
   - *Probabilitas mutasi*; perlu ditentukan besar kemungkinan terjadinya *mutasi* pada satu individu. Biasanya digunakan probabilitas sebesar 0,1.
   - Jumlah generasi; sebenarnya menunjukkan berapa kali iterasi yang dilakukan. Misalkan hendak dilakukan 10 kali iterasi, maka iterasi dihentikan apabila telah didapatkan generasi baru dari generasi ke-10.
   - Bobot untuk masing-masing kepentingan *Makespan* dan Total *Flowtime*. Bila tidak ditentukan, program dapat menghitungnya secara otomatis dimana akan dianggap keduanya memiliki bobot berimbang (tidak ada yang dipentingkan antara *makespan* dan total *flowtime*)
2. Solusi awal didapatkan dari algoritma *SPT (Shortest Processing Time)* dan jadwal-jadwal berikutnya dalam generasi awal ini dibangun dengan metode random
3. Untuk operator reproduksi, semua individu dalam satu generasi melakukan perkawinan dengan setiap individu lainnya. Jadi seandainya ada 4 individu, maka akan terjadi 6 kali perkawinan dan menghasilkan 12 *anak*.
4. Dihitung semua fungsi objektif anak yang dihasilkan, kemudian diurutkan dari nilai yang terbaik hingga terburuk. Sebanyak *x* anak dibiarkan hidup untuk menjadi generasi penerus berikutnya dan sisanya mati. Besar *x* sesuai dengan banyaknya individu dalam satu generasi awal yang telah ditentukan sebelumnya pada parameter awal. Seandainya ditetapkan bahwa dalam 1 populasi terdapat 4 individu, maka yang diluluskan menjadi generasi selanjutnya adalah 4 terbaik dari keseluruhan populasi.
5. Untuk pengawinannya akan dilakukan dengan metode *crossover*. Metode ini dimulai dengan mengambil dua buah angka random antara 1 hingga banyaknya *job*. Misalkan *crossover* diberlakukan

pada dua jadwal X dan Y berikut ini, dimana angka random yang muncul adalah 2 dan 5, maka :

X = 3-5-|4-6-1|-7-2
Y = 2-1-|7-5-6|-4-3

Setelah di-*crossover*, *job-job* dalam rentang angka random itu akan bertukar tempat secara bersesuaian posisinya, sehingga hasil *crossover* akan menjadi :

X' = 3-5-|7-5-6|-7-2
Y' = 2-1-|4-6-1|-4-3

Hasil *crossover* di atas menghasilkan X' dan Y', dimana X' : 3-5-7-5-6-7-2 dan Y' : 2-1-4-6-1-4-3. Dari hasil tersebut dapat dilihat bahwa masih ada gen yang sama dalam 1 kromosom, maka langkah yang harus dilakukan yaitu melakukan repair kromosom. Untuk kromosom X', hasil repair kromosom adalah 3-5-1-4-6-7-2 dan hasil repair kromosom Y' adalah 2-1-5-6-7-4-3.

X' = 3-5-|7-5-6|-7-2 → X' = 3-5-1-4-6-7-2
Y' = 2-1-|4-6-1|-4-3 → Y' = 2-1-5-6-7-4-3

6. Untuk perhitungan nilai objektif, karena ada lebih dari dua nilai yang diperbandingkan tentu tidak bisa menggunakan operator Δ seperti pada algoritma *simulated annealing*. Oleh karena itu dicari cara lain untuk melakukannya sebagai berikut :
   - Menghitung semua *makespan* (*Ms)* dan total *flowtime* (*F)* dari anak yang dihasilkan.
   - Menghitung *Msr* (makespan rata-rata) dan *Fr* (total flowtime rata-rata)
   - Menghitung *w1* (sesuai rumus 2.6) dan *w2* (sesuai rumus 2.7).
   - Apabila *bobot w1* dan *w2* telah ditentukan sendiri, maka tiga langkah di atas tidak perlu dilakukan
   - Berikutnya menghitung nilai objektif Y_i (sesuai rumus 2.8) = $w2\ Ms_i + w1\ F_i$. Konsep ini untuk menyeimbangkan bobot *makespan* (*Ms)* dan total *flowtime* (*F)*.
7. Setelah didapat semua nilai *Y* untuk semua anak barulah diurutkan dari yang terbaik (nilai *Y* yang terkecil) hingga terburuk dan diambil sebanyak jumlah individu per generasi yang telah ditentukan.
8. Operator *mutasi* diberlakukan setelah didapatkan individu baru. Caranya adalah dengan mengambil satu angka random antara 0 dan 1, dan jika angka ini lebih kecil atau sama dengan *probabilitas mutasi* yang ditentukan dari awal, maka individu ini

dikenai operator *mutasi*. Jika tidak, maka individu itu lolos tanpa perubahan apapun.

9. Bila terkena *operator mutasi*, langkah berikutnya adalah mengambil dua angka random misal *d1* dan *d2*, kemudian *job d1* dan *d2* pada individu tersebut bertukar posisi. Misal terdapat jadwal J = 3-7-1-5-2-6-4, d1 = 3 dan d2 = 5, maka *job* ke-3 akan bertukar posisi dengan *job* ke-5 menjadi J' = 5-7-1-3-2-6-4. Ulangi lagi mulai langkah ke-3 hingga syarat penghentian iterasi dipenuhi.
10. Jadwal terbaik adalah jadwal nomor 1 pada generasi terakhir.

3.3.3. Perancangan proses perbandingan algoritma genetika dan simulated annealing

Untuk proses pembandingan hasil akhir dari algoritma genetika maupun algoritma *simulated annealing* cukup dilakukan pembandingan pada satu jadwal yaitu jadwal terbaik dari masing-masing algoritma (seperti pada rumus 2.9). Jika $\Delta \leq 0$, maka *simulated annealing* (SA) lebih baik dari *genetic algorithm* (GA) dan sebaliknya jika $\Delta > 0$ maka *genetic algorithm* (GA) lebih baik dari *simulated annealing* (SA).

3.3.4. Perancangan perhitungan prosentase keunggulan antara algoritma genetika dan simulated annealing

Prosentase keunggulan antar algoritma adalah ukuran seberapa baik suatu jadwal mengungguli jadwal lainnya. Cara pencarian prosentase keunggulan tersebut adalah dari dua jadwal terbaik yang didapatkan, dihitung $\sum Ms$ (total *makespan*), Msr (*makespan* rata-rata), $\sum F$ (total *flowtime*), dan Fr (total *flowtime* rata-rata). Dengan menetapkan $w1$ (sesuai rumus 2.6), $w2$ (sesuai rumus 2.7) = $1 - w1$, dan Y_i (sesuai rumus 2.8) = $w2\ Ms_i + w1\ F_i$, maka didapat nilai Y untuk masing-masing algoritma. Jika *genetic algorithm* (GA) unggul, maka dapat dihitung sesuai dengan rumus 2.10 dan sebaliknya jika *simulated annealing* (SA) unggul, maka dapat dihitung sesuai dengan rumus 2.11.

3.3.5. Perancangan simulasi

Simulasi akan dilakukan dengan aturan sebagai berikut :

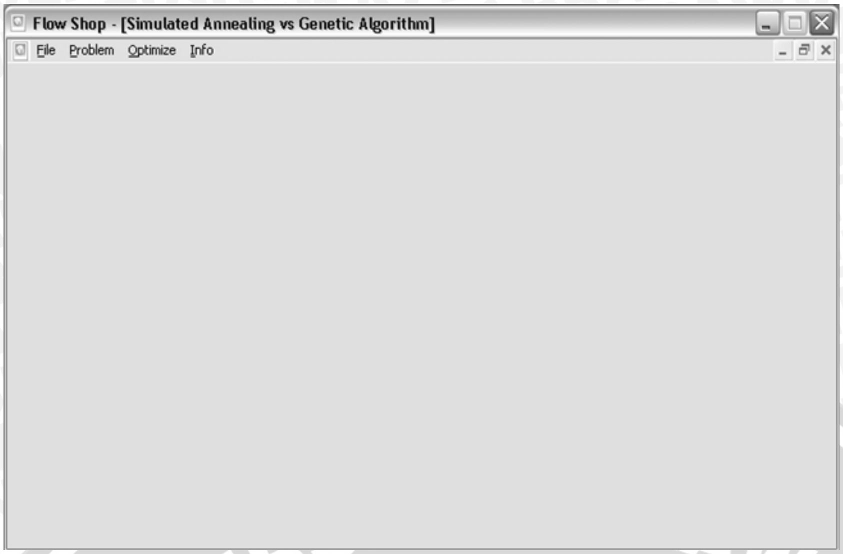
- Banyaknya *job* bervariasi (10,20,30,40, dan 50 *job*) pada tiap tahapnya.
- Banyaknya mesin bervariasi (5,10,15,20, dan 25 mesin) pada tiap tahapnya.
- Waktu proses untuk tiap job akan digunakan nilai random atau diisi secara manual oleh user.
- Jumlah problem tiap kombinasi adalah 5.
- Untuk algoritma *simulated annealing* ditentukan parameter awal sebagai berikut :
  - *Suhu awal* (T_0) = 1
  - *Faktor reduksi suhu* (r) = 0,9
  - *Replikasi* (R) = 3
- Jika pada parameter algoritma genetika terdapat 4 chromosome / individu dan 10 generasi, maka akan terjadi pencarian solusi tetangga (*neighborhood*) sebanyak 12 x 10 = 120 kali. Sehingga agar didapatkan pencarian solusi yang sama seperti algoritma genetika dengan 3 kali *replikasi*, maka diharapkan terjadi 40 kali penurunan suhu pada algoritma *simulated annealing* dengan penghitungan Ta (suhu akhir) = $T_0 \times (r)^{40} = 1 \times (0{,}9)^{40} = 0{,}01478$. Atau dengan kata lain bahwa suhu akhir didapatkan dengan cara perhitungan sebagai berikut :

  $$Ta = T_0 \times (r)^{(\text{anak x generasi}) / \text{Replikasi}}$$

3.4 Perancangan Interface

Perancangan interface dilakukan melalui beberapa tahapan pembuatan form, antara lain :

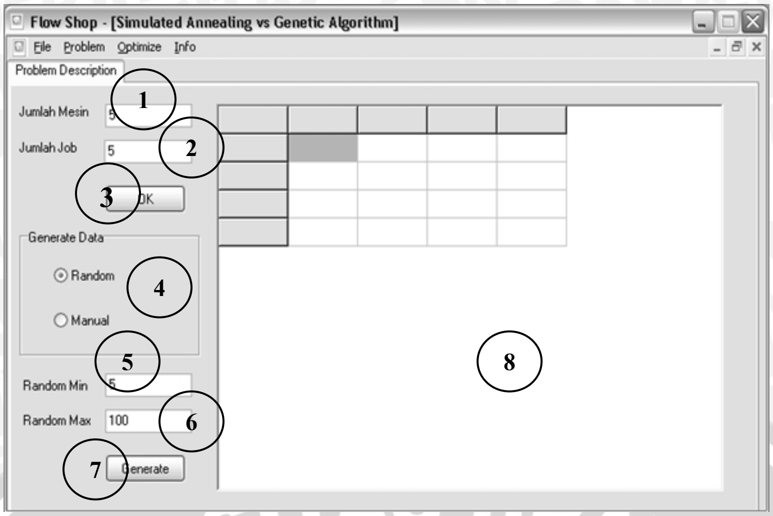
1. **Form Utama**



Gambar 3.3 Rancangan antarmuka form utama

Di dalam form utama terdapat menu-menu diantaranya menu *file*, *problem*, *optimize* dan *info*. Pada menu file terdapat sub menu ’new’ yang berfungsi untuk menampilkan form baru (memulai simulasi baru) dan sub menu ’exit’ yang berfungsi untuk menutup program. Pada menu problem terdapat sub menu ’data’ yang berfungsi menampilkan form input untuk mengisikan parameter problem. Pada menu optimize terdapat sub menu ’genetic algorithm’ yang berfungsi menampilkan perhitungan jadwal dengan algoritma genetika dan sub menu ’simulated annealing’ yang berfungsi menampilkan perhitungan jadwal dengan *simulated annealing*. Pada menu info terdapat sub menu ’result’ yang berfungsi untuk menampilkan hasil perbandingan perhitungan jadwal antara algoritma genetika dan *simulated annealing* dan sub menu ’graph’ yang berfungsi menampilkan hasil perbandingan processing time antara algoritma genetika dan *simulated annealing*.

2. Form Problem Description



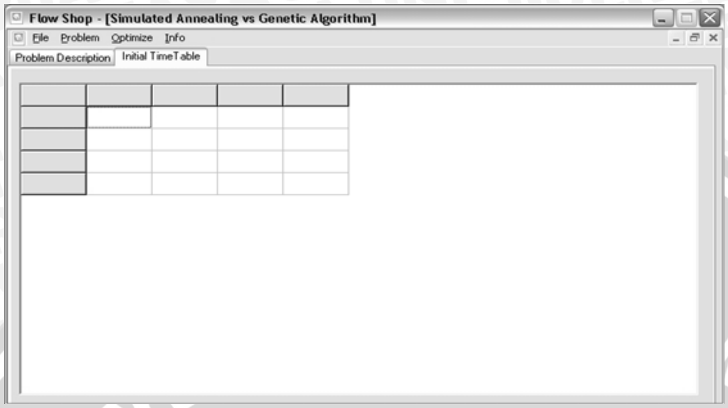
Gambar 3.4 Rancangan antarmuka form problem description

Di dalam form problem description terdapat beberapa tahapan yang harus diinputkan oleh user, yaitu :

1) User menginputkan jumlah mesin yang akan digunakan dalam penjadwalan.
2) User menginputkan jumlah job yang akan digunakan dalam penjadwalan.
3) Tombol 'OK' akan mengubah tampilan pada string grid (gambar 8), dimana jumlah job menggambarkan jumlah baris yang terbentuk dan jumlah mesin menggambarkan jumlah kolom yang terbentuk.
4) User akan memilih metode untuk mengisi waktu proses tiap jobnya pada string grid. Jika memilih 'random', maka komputer akan merandomkan angka secara otomatis dan jika memilih 'manual' user akan mengisikan sendiri berapa waktu proses yang dibutuhkan.
5) Jika user memilih generate data secara random, maka user harus mengisi nilai minimum dari angka random tersebut.
6) Sama halnya seperti gambar 5, jika user memilih generate data secara random, maka user harus mengisi nilai maksimum dari angka random tersebut.

7) Tombol ‘generate’ dimaksudkan untuk melakukan pencarian solusi awal dengan menggunakan algoritma SPT. Setelah tombol di-klik maka akan muncul form initial time table.
8) String grid 1 digunakan untuk menampilkan problem penjadwalan yang akan diselesaikan.

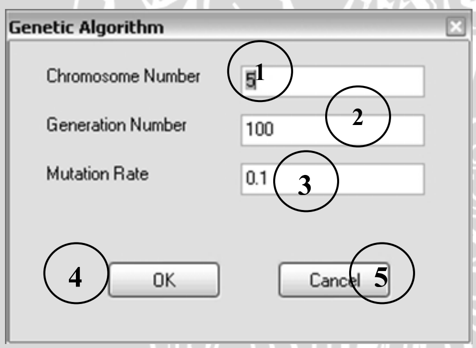
3. Form Initial Time Table Flow Shop



Gambar 3.5 Rancangan antarmuka form initial time table

Form ini digunakan untuk menampilkan hasil perhitungan solusi awal yang didapatkan dengan menggunakan algoritma SPT (Shortest Processing Time).

4. Form Genetic Parameter

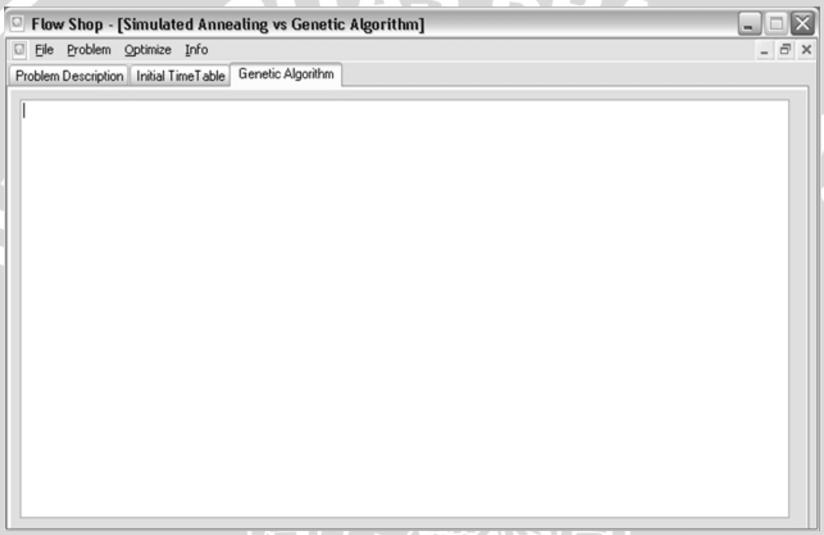


Gambar 3.6 Rancangan antarmuka form genetic parameter

Form ini digunakan user untuk menginputkan beberapa parameter awal yang digunakan pada algoritma genetika.

1) User menginputkan jumlah chromosome / individu dalam 1 generasi.
2) User menginputkan jumlah generasi untuk proses pengulangan (iterasi).
3) User menginputkan nilai probabilitas mutasi yang digunakan sebagai batas bagi chromosome / individu apakah perlu dimutasi atau tidak.
4) Tombol ‘OK’ menandakan bahwa user sudah yakin bahwa input parameter awal algoritma genetika sudah benar dan langsung diproses.
5) Tombol ‘cancel’ digunakan jika user tidak jadi menginputkan parameter awal algoritma genetika.

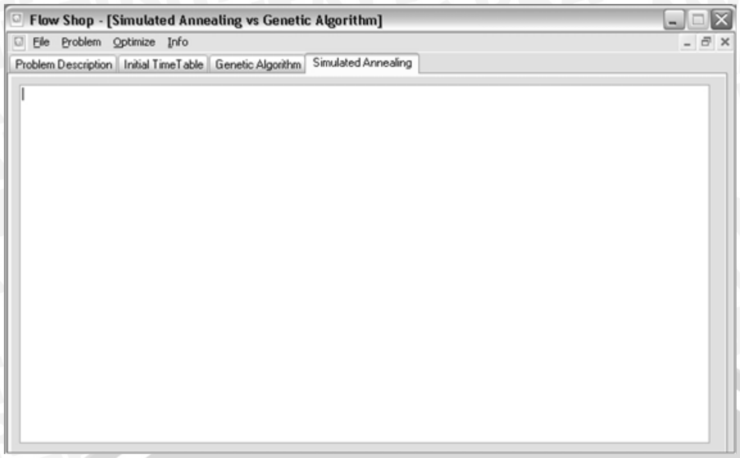
5. Form Genetic Calculate



Gambar 3.7 Rancangan antarmuka form genetic calculate

Form ini digunakan untuk menampilkan hasil perhitungan dari problem penjadwalan *flow shop* dengan menggunakan algoritma genetika.

6. **Form Simulated Annealing Calculate**



Gambar 3.8 Rancangan antarmuka form SA calculate

Form ini digunakan untuk menampilkan hasil perhitungan dari problem penjadwalan *flow shop* dengan menggunakan algoritma *simulated annealing*.

7. **Form Perbandingan**

Gambar 3.9 Rancangan antarmuka form perbandingan

Form ini digunakan untuk menampilkan hasil perbandingan problem penjadwalan *flow shop* dengan menggunakan algoritma *simulated annealing* dan algoritma genetika untuk beberapa problem dan simulasi.

8. Form Graph

Gambar 3.10 Rancangan antarmuka form graph

Form ini digunakan untuk menampilkan hasil perbandingan problem penjadwalan *flow shop* dengan menggunakan algoritma *simulated annealing* dan algoritma genetika dalam bentuk graph untuk beberapa problem dan simulasi.

BAB IV
IMPLEMENTASI DAN PEMBAHASAN

4.1 Lingkungan Implementasi

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

4.1.1 Lingkungan perangkat keras

Perangkat keras yang digunakan dalam pengembangan sistem flow shop scheduling ini adalah sebagai berikut :

1. Prosesor AMD Sempron 1.4 Ghz
2. Memori 512 MB
3. Harddisk dengan kapasitas 80 GB
4. Monitor 15”
5. Keyboard
6. Mouse

4.1.2 Lingkungan perangkat lunak

Perangkat lunak yang digunakan dalam pengembangan sistem flow shop scheduling ini adalah :

1. Sistem Operasi Windows XP
2. Borland Delphi 6

4.2 Implementasi Program

Berdasarkan analisa dan perancangan proses yang terdapat pada subbab 3.3, maka pada subbab ini akan dijelaskan implementasi proses-proses tersebut.

4.2.1 Implementasi Awal (Problem Setting)

Tahap awal dari flow shop scheduling ini adalah men-setting input jumlah mesin dan *job*. Selain itu dilakukan juga pembangkitan bilangan acak baik secara random ataupun manual. Berikut ini adalah listing program untuk melakukan generate data (problem setting) dan input jumlah mesin dan *job*.

```
procedure TFormSAGA.ButtonOKClick(Sender: TObject);
var i,j : integer;
begin
   ClearStringGrid(SG1,1);
   ClearStringGrid(SGBanding,2);

   SG1.RowCount := StrToInt(EditJob.Text) + 1;
   SG1.ColCount := StrToInt(EditMesin.Text) + 1;
   for i := 1 to SG1.RowCount-1 do
      SG1.Cells[0,i] := 'Job ' + IntToStr(i);
   for j := 1 to SG1.ColCount-1 do
      SG1.Cells[j,0] := 'Mesin ' + IntToStr(j);
   for j := 1 to SG1.ColCount-1 do
   begin
     for i := 1 to SG1.RowCount-1 do
        SG1.Cells[j,i] := '';
   end;

   nMesin := SG1.ColCount-1;
   nJob := SG1.RowCount-1;

   for i := 0 to 6 do
     SGBanding.Cells[i,2] := '';
   score_SA := 0;
   score_GA := 0;
   performa_GA_rata2 := 0;
   performa_SA_rata2 := 0;
   waktuProses_SA := 0;
   waktuProses_GA := 0;
end;
```

Pada prosedur diatas, akan dihasilkan string grid dengan jumlah kolom dan baris yang sesuai dengan jumlah job dan mesin yang diinputkan, dimana kolom mengindikasikan jumlah mesin dan baris mengindikasikan jumlah job. Sedangkan untuk generate data dapat dilihat pada listing berikut ini.

```
procedure TFormSAGA.ButtonGenerateClick(Sender: TObject);
var
  a,b,i,j,rand,total : integer;
begin
   ClearStringGrid(SG2,1);
   ClearStringGrid(SG3,2);
   SG2.RowCount := StrToInt(EditJob.Text) + 1;
   SG2.ColCount := StrToInt(EditMesin.Text) + 2;
   SG3.RowCount := StrToInt(EditJob.Text) + 2;
   SG3.ColCount := (SG1.ColCount-1) * 2 + 1;
   for i := 1 to SG2.RowCount-1 do
      SG2.Cells[0,i] := 'Job ' + IntToStr(i);
   for j := 1 to SG2.ColCount-1 do
      SG2.Cells[j,0] := 'Mesin ' + IntToStr(j);
   SG2.Cells[SG2.ColCount-1,0] := 'Total';
   i := 1;
   for j := 1 to SG3.ColCount-1 do
   begin
      if (j mod 2) <> 0 then
      begin
         SG3.Cells[j,0] := 'Mesin ' + IntToStr(i);
         SG3.Cells[j,1] := 'Start ';
         inc(i);
      end
      else
         SG3.Cells[j,1] := 'End ';
   end;
   if RBRandom.Checked = true then
   begin
      Randomize;
      a := StrToInt(editrandommin.Text);
      b := StrToInt(editrandommax.Text);
      for i := 1 to SG1.RowCount-1 do
      begin
         for j := 1 to SG1.ColCount-1 do
         begin
            rand := Random((b-a)+1) + a;
            SG1.Cells[j,i] := IntToStr(rand);
            SG2.Cells[j,i] := SG1.Cells[j,i];
         end;
      end;
   end
   else if RBManual.Checked = true then
   begin
      for i := 1 to SG1.RowCount-1 do
      begin
         for j := 1 to SG1.ColCount-1 do
         begin
            SG2.Cells[j,i] := SG1.Cells[j,i];
         end;
      end;
   end;
```

```
    for i := 1 to SG2.RowCount-1 do
    begin
       total := 0;
       for j := 1 to SG2.ColCount-2 do
       begin
          total := total + StrToInt(SG2.Cells[j,i]);
          SG2.Cells[Sg2.ColCount-1,i] := IntToStr(total);
       end;
    end;
    TabInitialTimeTable.TabVisible := True;

    //buat fungsi evaluasi
    nMesin := SG1.ColCount-1;
    nJob := SG1.RowCount-1;
    SetLength(data,nMesin,nJob);
    BacaData(SG1, data);
    fungsi := TFungsiEvaluasi.Create(data,nJob,nMesin);
end;
```

Pada prosedur di atas, dilakukan *generate data* untuk membangkitkan bilangan acak, baik secara random maupun manual. Data tersebut juga disimpan pada SG2 yang merupakan data initial time table. Hasil *generate data* akan disimpan pada class TFungsiEvaluasi yang menjadi problem awal proses flow shop scheduling.

4.2.2 Implementasi *SPT (Shortest Processing Time)*

Setelah dilakukan proses *generate data*, maka langkah selanjutnya adalah melakukan penjadwalan sebagai solusi awal pada flow shop. Data pada SG2 diurutkan terlebih dahulu dengan menggunakan algoritma SPT (*Shortest Processing Time*). Data yang telah diurutkan akan dihitung nilai *makespan* dan total *flowtime*nya sebagai solusi awal dari penjadwalan. Proses tersebut dapat dilihat pada listing program di bawah ini.

```
procedure TFormSAGA.ButtonSortClick(Sender: TObject);
var i,j,k,m,tmp : integer;
    urutan : TList;
begin
   //urutan hasil SPT
   SetLength(urutan,SG1.RowCount-1);
   for i := 0 to High(urutan) do
   begin
     urutan[i] := i+1;
   end;
   for i := 1 to SG2.RowCount-1 do
   begin
      for j := i+1 to SG2.RowCount-1 do
      begin
         if (StrToInt(SG2.Cells[SG2.colCount-1,j])) <
             (StrToInt(SG2.Cells[SG2.colCount-1,i])) then
         begin
            //tukar posisi job pada SG2
            tukar(i,j);
            //tukar urutan
            tmp := urutan[i-1];
            urutan[i-1] := urutan[j-1];
            urutan[j-1] := tmp;
         end;
      end;
   end;
   //simpan urutannya disini
   fungsi.SetSPT(urutan);

   for i := 2 to SG3.RowCount-1 do
   begin
      SG3.Cells[0,i] := SG2.Cells[0,i-1];
      for j := 1 to SG3.ColCount-1 do
         SG3.Cells[j,i] := '0';
   end;

   k := 1; //index buat baris SG2
   for i := 2 to SG3.RowCount-1 do
   begin
      j := 2;
      m := 1; //index buat kolom SG2
      while (j < SG3.ColCount) do
      begin
         if (k = 1) then
         begin
            if (m = 1) then
            begin
               SG3.Cells[j-1,i] := '0'; // start
               SG3.Cells[j,i] := SG2.Cells[m,k]; //end
            end
            else
            begin
               SG3.Cells[j-1,i] := SG3.Cells[j-2,i];
               SG3.Cells[j,i] := SG2.Cells[m,k]; //end
```

```
            end
            else
            begin
               SG3.Cells[j-1,i] := SG3.Cells[j-2,i];
               SG3.Cells[j,i] :=
                    IntToStr(StrToInt(SG3.Cells[j-1,i]) +
                    StrToInt(SG2.Cells[m,k]));
            end;
         end
         else
         begin
            if (m = 1) then
            begin
               SG3.Cells[j-1,i] := SG3.Cells[j,i-1];
               SG3.Cells[j,i] :=
                    IntToStr(StrToInt(SG3.Cells[j-1,i]) +
                    StrToInt(SG2.Cells[m,k]));
            end
            else
            begin
               if (StrToInt(SG3.Cells[j-2,i]) >
                   StrToInt(SG3.Cells[j,i-1])) then
               begin
                  SG3.Cells[j-1,i] := SG3.Cells[j-2,i];
                  SG3.Cells[j,i] :=
                     IntToStr(StrToInt(SG3.Cells[j-1,i]) +
                     StrToInt(SG2.Cells[m,k]));
               end
               else if (StrToInt(SG3.Cells[j-2,i]) <=
                        StrToInt(SG3.Cells[j,i-1])) then
               begin
                  SG3.Cells[j-1,i] := SG3.Cells[j,i-1];
                  SG3.Cells[j,i] :=
                      IntToStr(StrToInt(SG3.Cells[j-1,i]) +
                      StrToInt(SG2.Cells[m,k]));
               end;
            end;
         end;
         m := m+1;
         j := j+2;
      end;
      k := k+1;
   end;
end;
```

4.2.3 Implementasi Flow Shop dengan Algoritma Genetika

Tahapan ini dilakukan setelah penjadwalan solusi awal telah terbentuk. Proses penghitungan pada algoritma genetika ini dibagi menjadi beberapa langkah yaitu :

4.2.3.1. Pembentukan Kromosom

Pada langkah ini, terlebih dahulu ditentukan ukuran populasi yaitu jumlah kromosom pada populasi tersebut, mutation rate (probabilitas mutasi), dan panjang dari masing-masing kromosom. Sebagai inisialisasi, kromosom ke-1 pada populasi diisi dengan urutan hasil SPT yang telah didapat pada proses sebelumnya. Kromosom-kromosom lain dihasilkan dari data random. Lebih jelasnya dapat dilihat pada listing program berikut ini.

```
constructor TPopulasi.Create(fungsi: TFungsiEvaluasi;
ukuranPopulasi: Integer; nJob, nMesin: Integer; mutationRate:
Real);
var
  i,k: Integer; str : String;
  tmp: TList;
begin
  Self.ukuranPopulasi := ukuranPopulasi;
  Self.panjangKromosom := nJob;

  SetLength(Self.populasi, Self.ukuranPopulasi);

  Self.fungsiEvaluasi := fungsi;

  //untuk kromosom ke 1 gunakan SPT
  Self.populasi[0] := TKromosom.Create(Self.panjangKromosom);

  tmp := Self.fungsiEvaluasi.SPT;

  for i := 0 to Self.panjangKromosom-1 do
     Self.populasi[0].kromosom[i] := tmp[i];

  //untuk kromosom selanjutnya gunakan cara random
  for i := 1 to Self.ukuranPopulasi-1 do
  begin
    Self.populasi[i] :=
         TKromosom.Create(Self.panjangKromosom);
  end;
end;
```

Untuk panjang dari kromosom ke-1 digunakan hasil dari SPT, yaitu dengan cara memanggil fungsiEvaluasi.SPT dan mengkopikan nilai (urutan job) pada kromosom ke-1 tersebut. Untuk kromosom-

kromosom selanjutnya digunakan cara random dengan cara memanggil fungsi TKromosom.Create. Listing programnya dapat dijelaskan seperti di bawah ini.

```
constructor TKromosom.Create(panjangKromosom: Integer);
begin
  Self.panjangKromosom := panjangKromosom; //mengambil
                         panjang kromosom pada populasi
  Self.fitness := 0;
  Self.makespan := 0;
  Self.flowTime := 0;
  Self.GenerateKromosom;
end;

procedure TKromosom.GenerateKromosom;
var
  i: Integer;
  value: Integer;
begin
  Setlength(Self.kromosom,Self.panjangKromosom);

  for i := 0 to panjangKromosom-1 do
  begin
    Self.kromosom[i] := 0;
  end;

  i := 0;
  //Randomize;
  while (i < Self.panjangKromosom) do
  begin
    value := Random(Self.panjangKromosom) + 1;
    if (CekKromosom(value) = false) then
    begin
      Self.kromosom[i] := value;
      Inc(i);
    end;
  end;
end;
```

4.2.3.2. Crossover Kromosom

Pada langkah ini, tiap kromosom yang terbentuk akan saling melakukan *crossover*. Misal terdapat 3 *parent* maka akan terjadi 3 kali *crossover* yaitu p1 dengan p2, p1 dengan p3 dan p2 dengan p3 dan menghasilkan masing-masing 2 *child*. Jadi total hasil dari *crossover* 3 *parent* adalah 6 *child*. Hasil dari crossover (*child*) digabungkan dengan *parent* dalam satu populasi dan dilakukan proses evaluasi untuk mendapatkan nilai *makespan* dan total *flowtime*. Listing programnya dapat dilihat seperti di bawah ini.

```
procedure TPopulasi.Crossover;
var
  i, k, m: Integer;
  c1, c2: TKromosom;
  ukuranPop: Integer;
begin
  ukuranPop := Self.ukuranPopulasi;

  Randomize;

  for i := 0 to Self.ukuranPopulasi-2 do
  begin
    for k := i + 1 to Self.ukuranPopulasi-1 do
    begin
      c1 := TKromosom.Create(Self.panjangKromosom);
      c2 := TKromosom.Create(Self.panjangKromosom);
      for m := 0 to Self.panjangKromosom-1 do
        c1.kromosom[m] := Self.populasi[i].kromosom[m];
      c1.makespan := Self.populasi[i].makespan;
      c1.flowTime := Self.populasi[i].flowTime;
      c1.fitness := Self.populasi[i].fitness;

      for m := 0 to Self.panjangKromosom-1 do
        c2.kromosom[m] := Self.populasi[k].kromosom[m];
      c2.makespan := Self.populasi[k].makespan;
      c2.flowTime := Self.populasi[k].flowTime;
      c2.fitness := Self.populasi[k].fitness;

      c1.Crossover(c2);

      fungsiEvaluasi.Evaluasi(c1.kromosom);
      c1.makespan := Self.fungsiEvaluasi.MakespanValue;
      c1.flowTime := Self.fungsiEvaluasi.FlowTimeValue;

      fungsiEvaluasi.Evaluasi(c2.kromosom);
      c2.makespan := Self.fungsiEvaluasi.MakespanValue;
      c2.flowTime := Self.fungsiEvaluasi.FlowTimeValue;

      ukuranPop := ukuranPop + 2;
      SetLength(Self.populasi,ukuranPop);

      Self.populasi[ukuranPop-2] := c1;
      Self.populasi[ukuranPop-1] := c2;
    end;
  end;
  Self.EvaluasiFitness;
end;
```

Dalam listing program tersebut juga dipanggil fungsi TKromosom.Crossover, dimana perhitungan crossover antar kromosom dapat dilihat pada listing program berikut ini.

```
procedure TKromosom.Crossover(var c: TKromosom);
var
  k,posisiCrossover: Integer;
  str: String;
begin
  //posisi gen yang akan di-crossover
  posisiCrossover := Random(Self.panjangKromosom-1);
  if (posisiCrossover = 0) then
     posisiCrossover := 1;

  //ubah kromosom disini
  UbahKromosom(Self.kromosom,c.kromosom,posisiCrossover);

  //repair
  RepairKromosom(Self.kromosom,c.kromosom);
end;
```

Prosedur di atas menjelaskan bagaimana crossover antar kromosom diproses. Hal pertama yang dilakukan adalah dengan menentukan posisi gen yang akan ditukar. Kemudian panggil prosedur UbahKromosom berdasarkan posisi gen yang telah ditentukan untuk meng-crossover kromosom tersebut. Dan langkah terakhir adalah memanggil prosedur RepairKromosom, jika ada gen yang sama dari 2 kromosom yang di-crossover.

4.2.3.3. Seleksi Kromosom

Pada langkah ini, kromosom-kromosom yang telah di-crossover akan diseleksi dan diambil sebanyak jumlah kromosom awal. Penyeleksian dilakukan dengan cara sorting (mengurutkan) kromosom sesuai dengan nilai fitness terendah. Sedangkan sisa kromosom yang lain (yang tidak lolos seleksi) tidak akan dipertahankan / dihapus. Listing programnya dapat dilihat seperti di bawah ini.

```
procedure TPopulasi.SeleksiKromosom;
var
  ukuranPop: Integer;
  i: Integer;
begin
  ukuranPop := High(Self.populasi);
  Sorting(ukuranPop);
  for i := Self.ukuranPopulasi to ukuranPop do
  begin
    Self.populasi[i].Free;
  end;
  SetLength(Self.populasi,Self.ukuranPopulasi);
end;
```

```
procedure TPopulasi.Sorting(i: Integer);
var
  k, m: Integer;
  tmpKromosom: TKromosom;
begin
   //bubble sort
   for k := 0 to i-1 do
   begin
     for m := k+1 to i do
     begin
       if (Self.populasi[m].fitness <
           Self.populasi[k].fitness) then
       begin
         tmpKromosom := Self.populasi[k];
         Self.populasi[k] := Self.populasi[m];
         Self.populasi[m] := tmpKromosom;
       end;
     end;
   end;
end;
```

4.2.3.4. Mutasi Kromosom

Setelah proses seleksi dilakukan, maka langkah selanjutnya adalah proses mutasi pada kromosom terpilih. Tahap pertama adalah membangkitkan bilangan acak antara 0 sampai 1. Jika nilai random yang muncul kurang dari *mutation rate* yang telah ditentukan maka kromosom tersebut akan dimutasi. Kemudian setelah didapatkan kromosom hasil mutasi, maka dilakukan lagi evaluasi untuk mendapatkan nilai *makespan* dan total *flowtime* yang baru. Lebih jelasnya dapat dilihat pada listing berikut ini.

```
procedure TPopulasi.Mutasi;
var
  i: Integer;
  val: Real;
begin
  i := 0;
  Randomize;
  while (i <= Self.ukuranPopulasi-1) do
  begin
    val := Random(1000)/1000;
    if (val < Self.mutationRate) then
    begin
      Self.populasi[i].Mutasi;
      fungsiEvaluasi.Evaluasi(Self.populasi[i].kromosom);
      Self.populasi[i].makespan :=
                       Self.fungsiEvaluasi.MakespanValue;
      Self.populasi[i].flowTime :=
                       Self.fungsiEvaluasi.FlowTimeValue;
    end;
```

```
    i := i + 1;
  end;
  Self.EvaluasiFitness;
end;
```

Proses mutasi pada kromosom hampir sama dengan proses crossover, dimana terlebih dahulu mencari 2 posisi gen yang akan dimutasi. Jika posisi mutasi 1 sama dengan posisi mutasi 2, maka akan dicari posisi mutasi yang baru secara random. Lebih jelasnya dapat dilihat pada listing berikut ini.

```
procedure TKromosom.Mutasi;
var
  posisiMutasi1, posisiMutasi2: Integer;
  tmp: Integer;
begin
  posisiMutasi1 := Random(Self.panjangKromosom);
  posisiMutasi2 := Random(Self.panjangKromosom);
  while (posisiMutasi1 = posisiMutasi2) do
  begin
    posisiMutasi2 := Random(Self.panjangKromosom);
  end;

  tmp := Self.kromosom[posisiMutasi1];
  Self.kromosom[posisiMutasi1] :=
       Self.kromosom[posisiMutasi2];
  Self.kromosom[posisiMutasi2] := tmp;
end;
```

4.2.4 Implementasi Flow Shop dengan Simulated Annealing

Proses perhitungan flow shop dengan algoritma *simulated annealing* dilakukan dengan beberapa tahapan / langkah yaitu :

4.2.4.1. Menentukan Parameter Awal

Pada langkah ini ditentukan parameter awal yang harus diinputkan antara lain besar temperatur awal, replikasi, dan faktor reduksi suhu. Sedangkan temperatur akhir ditentukan melalui beberapa perhitungan agar didapatkan jumlah iterasi yang sama dengan algoritma genetika. Listing program dapat dilihat seperti berikut ini

```
  suhuAwal := 1;
  replikasi := 2;
  reduksiSuhu := 0.9;
  suhuAkhir := suhuAwal *
  Power(reduksiSuhu,(jumlahKromosom*jumlahGenerasi/replikasi));
  simulatedAnnealing := TSimulatedAnnealing.Create(fungsi,
nJob, nMesin, replikasi, suhuAwal, suhuAkhir, reduksiSuhu);
```

Untuk menyimpan parameter input, maka dipanggil fungsi TSimulatedAnnealing.Create. Listing program dapat dilihat seperti berikut ini.

```
constructor TSimulatedAnnealing.Create(fungsi:
TFungsiEvaluasi;
        nJob, nMesin, replikasi: Integer; suhuAwal,
suhuAkhir, reduksi: Real);
begin
  Self.faktorReplikasi := replikasi;
  Self.temperaturAwal := suhuAwal;
  Self.temperaturAkhir := suhuAkhir;
  Self.faktorReduksiTemperatur := reduksi;

  Self.nJob := nJob;
  Self.nMesin := nMesin;
  Self.fungsiEvaluasi := fungsi;
end;
```

4.2.4.2. Membentuk Solusi Awal

Setelah parameter awal diinputkan langkah selanjutnya adalah membentuk solusi awal. Sebagai inisialisasi, maka solusi awal pada simulated annealing diisi dengan urutan job hasil SPT yang telah didapat pada proses sebelumnya. Listing program dapat dilihat seperti berikut ini.

```
SetLength(Self.solusiAwal.urutanJob, Self.nJob);
SetLength(Self.solusiBaru.urutanJob, Self.nJob);

//langkah pertama masukkan hasil SPT ke solusi Awal
tmp := fungsiEvaluasi.SPT;

for i := 0 to Self.nJob-1 do
  Self.solusiAwal.urutanJob[i] := tmp[i];

Self.fungsiEvaluasi.Evaluasi(Self.solusiAwal.urutanJob);
Self.solusiAwal.makespan :=
                     Self.fungsiEvaluasi.MakespanValue;
Self.solusiAwal.flowTime :=
                     Self.fungsiEvaluasi.FlowTimeValue;
```

4.2.4.3. Perhitungan Simulated Annealing

Proses penjadwalan flow shop dengan simulated annealing seperti pada subbab 3.3.1 dapat digambarkan dengan listing di bawah ini.

```
Randomize;
j := 0;

while (Self.temperaturAwal > Self.temperaturAkhir) do
begin
  //Replikasi
  for i := 1 to Self.faktorReplikasi do
  begin
    posisiTukar1 := Random(Self.nJob);
    posisiTukar2 := Random(Self.nJob);

    //pastikan posisi job yang ditukar tidak akan sama
    while (posisiTukar1 = posisiTukar2) do
    begin
      posisiTukar2 := Random(Self.nJob);
    end;

    //pertama kopikan dahulu solusi awal ke solusi baru
    for k := 0 to Self.nJob-1 do
       Self.solusiBaru.urutanJob[k] :=
                       Self.solusiAwal.urutanJob[k];
    Self.solusiBaru.makespan := Self.solusiAwal.makespan;
    Self.solusiBaru.flowTime := Self.solusiAwal.flowTime;

    //lakukan pertukaran posisi job pada urutan job di
      solusi baru
    tmpJob := Self.solusiBaru.urutanJob[posisiTukar1];
    Self.solusiBaru.urutanJob[posisiTukar1] :=
               Self.solusiBaru.urutanJob[posisiTukar2];
    Self.solusiBaru.urutanJob[posisiTukar2] := tmpJob;

    //lakukan evaluasi untuk mencari makespan dan flow time
      dari solusi baru
   Self.fungsiEvaluasi.Evaluasi(Self.solusiBaru.urutanJob);
   Self.solusiBaru.makespan :=
                       Self.fungsiEvaluasi.MakespanValue;
   Self.solusiBaru.flowTime :=
                       Self.fungsiEvaluasi.FlowTimeValue;

    //hitung nilai e1 berdasarkan solusi awal dan e2
      berdasarkan solusi baru
    Self.e1 := Self.HitungNilaiE(solusiAwal, solusiBaru);
    Self.e2 := Self.HitungNilaiE(solusiBaru, solusiAwal);

    Self.delta_E := Self.e2 - Self.e1;
    Self.probabilitas := 0;
    Self.px := 0;

    //terima solusi baru jika Delta E <= 0
    if (Self.delta_E <= 0) then
```

```
        Self.solusiAwal := Self.solusiBaru
      else
      begin
        //lakukan perhitungan probabilitas
        Self.probabilitas := Exp((-1) * Self.delta_E /
                                  Self.temperaturAwal);
        //generate nilai random untuk px
        Self.px := Random(1000)/1000;
        //terima jika px < prob
        if (Self.px < Self.probabilitas) then
          Self.solusiAwal := Self.solusiBaru;
      end;

      inc(j);
    end;

    //reduksi suhunya
    Self.temperaturAwal := Self.temperaturAwal *
                           Self.faktorReduksiTemperatur;
  end;
```

Untuk perhitungan nilai e untuk setiap solusi digunakan fungsi HitungNilaiE seperti listing program di bawah ini

```
function TSimulatedAnnealing.HitungNilaiE(solusi1, solusi2:
TSolusi) : Real;
var
  hasil: Real;
begin
  //hitung dari makespan dulu
  hasil := 0.5*((solusi1.makespan -
                Min(solusi1.makespan,solusi2.makespan)) /
                Min(solusi1.makespan,solusi2.makespan));
  //baru melalui flow time
  hasil := hasil + 0.5*((solusi1.flowTime -
                Min(solusi1.flowTime,solusi2.flowTime)) /
                Min(solusi1.flowTime,solusi2.flowTime));
  Result := hasil;
end;
```

4.2.5 Implementasi Proses Perbandingan dan Perhitungan Prosentase Keunggulan antara Algoritma Genetika dan Simulated Annealing

Proses perbandingan kinerja dan perhitungan prosentase keunggulan antara algoritma genetika dan *simulated annealing* dilakukan setelah perhitungan flow shop dari kedua algoritma tersebut selesai. Seperti yang telah dijelaskna pada subbab 3.3.3 dan subbab 3.3.4, maka perbandingan dan perhitungan prosentase keunggulan antara algoritma genetika dan *simulated annealing* dapat dilihat pada listing program berikut ini.

```
MsGA := StrToFloat(SGBanding.Cells[1,SGBanding.RowCount-1]);
FGA  := StrToFloat(SGBanding.Cells[2,SGBanding.RowCount-1]);
MsSA := StrToFloat(SGBanding.Cells[3,SGBanding.RowCount-1]);
FSA  := StrToFloat(SGBanding.Cells[4,SGBanding.RowCount-1]);

e1 := 0.5 * ((MsGA - min(MsGA, MsSA)) / min(MsGA, MsSA)) +
      0.5 * ((FGA - min(FGA, FSA)) / min(FGA, FSA));
e2 := 0.5 * ((MsSA - min(MsGA, MsSA)) / min(MsGA, MsSA)) +
      0.5 * ((FSA - min(FGA, FSA)) / min(FGA, FSA));
delta := e2 - e1;

w1 := ((MsGA + MsSA)/2) / ((FGA + FSA)/2);
w2 := 1 - w1;
YGA := (MsGA * w2) + (FGA * w1);
YSA := (MsSA * w2) + (FSA * w1);

if (delta <= 0) then
begin
  SGBanding.Cells[5,SGBanding.RowCount-1] := 'SA';
  inc(score_SA);
  prosentase_SA := abs((YSA - YGA)/YSA) * 100;
  prosentase_SA_rata2 := prosentase_SA_rata2 +
prosentase_SA;
  SGBanding.Cells[6,SGBanding.RowCount-1] :=
     FloatToStr(prosentase_SA);
end
else
begin
   SGBanding.Cells[5,SGBanding.RowCount-1] := 'GA';
   inc(score_GA);
   prosentase_GA := abs((YGA - YSA)/YGA) * 100;
   prosentase_GA_rata2 := prosentase_GA_rata2+
prosentase_GA;
   SGBanding.Cells[6,SGBanding.RowCount-1] :=
      FloatToStr(prosentase_GA);
end;
```

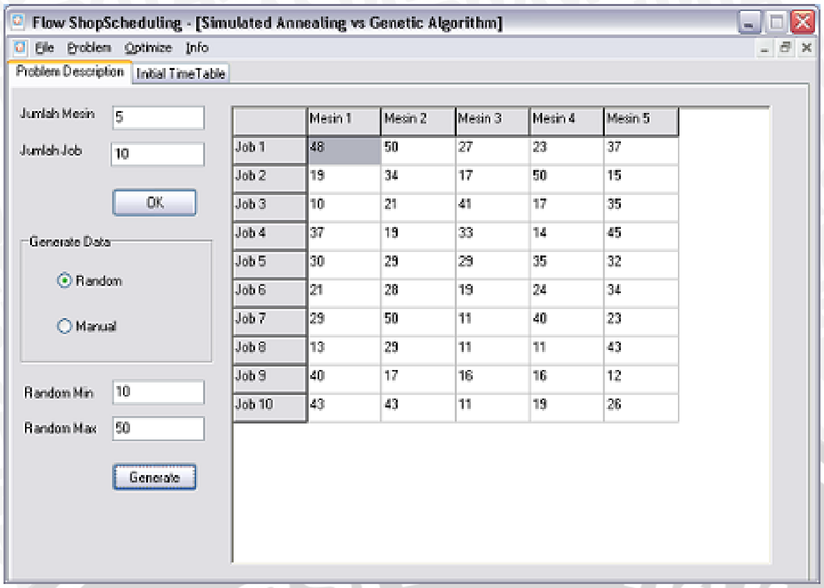
4.2.6 Implementasi Proses Simulasi

Proses simulasi menggunakan aturan sesuai dengan subbab 3.3.5 dimana jumlah job maupun mesin bervariasi. Sehingga total dari kombinasi job dan mesin tersebut adalah sebanyak 25. Pada simulasi kali ini akan dilakukan percobaan masing-masing kombinasi sebanyak 15 kali perulangan agar dapat diketahui algoritma mana yang lebih unggul pada tiap-tiap percobaan. Untuk waktu proses pada masing-masing job akan dibangkitkan nilai / bilangan acak antara 10 hingga 50. Lebih jelasnya dapat dilihat pada listing program berikut ini.

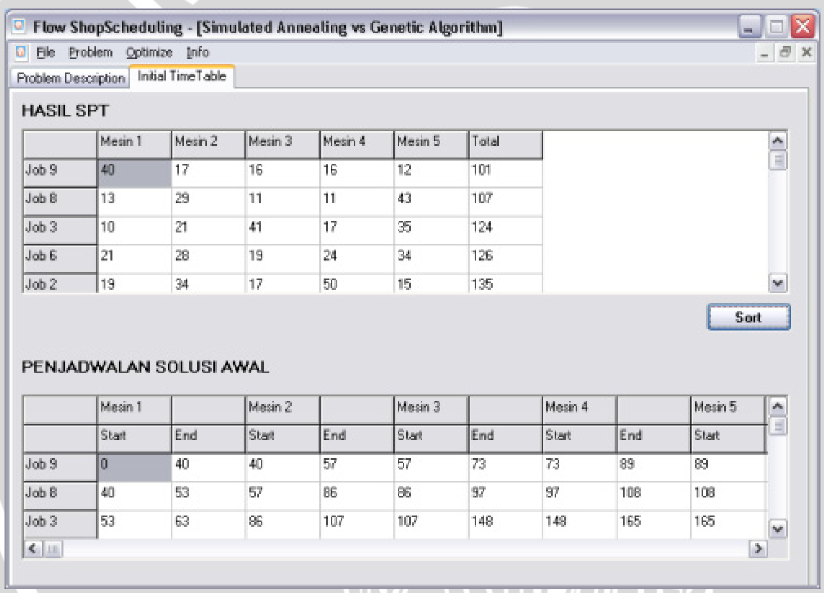
```
if SGBanding.RowCount < 17 then
   SGBanding.RowCount := SGBanding.RowCount + 1
else
begin
   SGHasilSimulasi.Cells[0,SGHasilSimulasi.RowCount-1] :=
             IntToStr(SGHasilSimulasi.RowCount-2);
   SGHasilSimulasi.Cells[1,SGHasilSimulasi.RowCount-1] :=
             IntToStr(nJob);
   SGHasilSimulasi.Cells[2,SGHasilSimulasi.RowCount-1] :=
             IntToStr(nMesin);
   SGHasilSimulasi.Cells[3,SGHasilSimulasi.RowCount-1] :=
             IntToStr(score_GA);
   SGHasilSimulasi.Cells[4,SGHasilSimulasi.RowCount-1] :=
             IntToStr(score_SA);

   if score_GA = 0 then
   begin
     SGHasilSimulasi.Cells[5,SGHasilSimulasi.RowCount-1] :=
              '0';
     SGHasilSimulasi.Cells[6,SGHasilSimulasi.RowCount-1] :=
              FloatToStr(performa_SA_rata2/score_SA);
   end
   else if score_SA = 0 then
   begin
     SGHasilSimulasi.Cells[6,SGHasilSimulasi.RowCount-1] :=
              '0';
     SGHasilSimulasi.Cells[5,SGHasilSimulasi.RowCount-1] :=
              FloatToStr(performa_GA_rata2/score_GA);
   end
   else
   begin
     SGHasilSimulasi.Cells[5,SGHasilSimulasi.RowCount-1] :=
             FloatToStr(performa_GA_rata2/score_GA);
     SGHasilSimulasi.Cells[6,SGHasilSimulasi.RowCount-1] :=
             FloatToStr(performa_SA_rata2/score_SA);
   end;
   SGHasilSimulasi.Cells[7,SGHasilSimulasi.RowCount-1] :=
           FloatToStr(waktuProses_GA/15);
   SGHasilSimulasi.Cells[8,SGHasilSimulasi.RowCount-1] :=
           FloatToStr(waktuProses_SA/15);
   SGHasilSimulasi.RowCount := SGHasilSimulasi.RowCount + 1;
end;
```

4.3 Implementasi Antarmuka

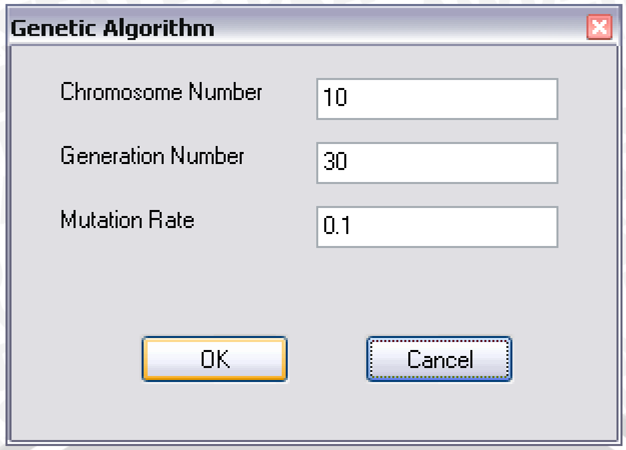
Berdasarkan rancangan antarmuka pada subbab 3.4 maka dihasilkan antarmuka berikut ini :



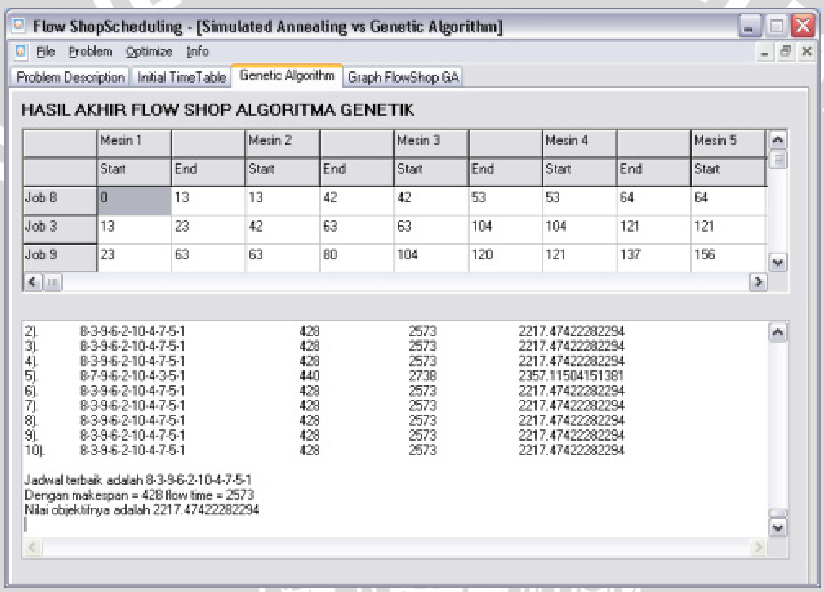
Gambar 4.1 Antarmuka *Problem Description*



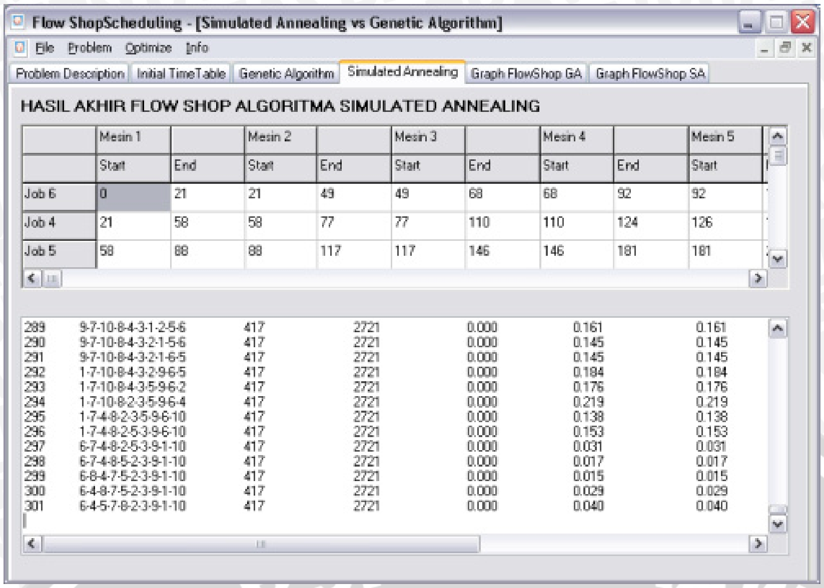
Gambar 4.2 Antarmuka *Initial Time Table*



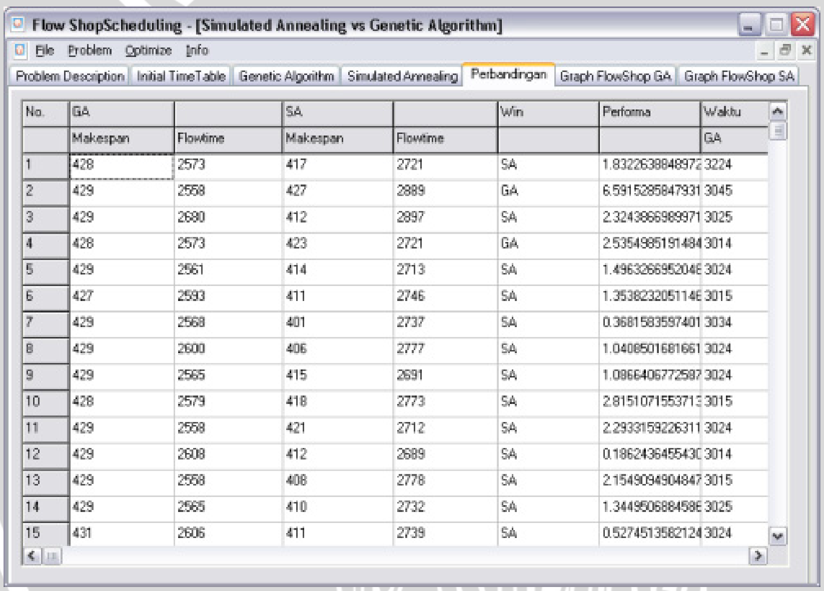
Gambar 4.3 Antarmuka *Genetic Parameter*



Gambar 4.4 Antarmuka *Genetic Calculate*



Gambar 4.5 Antarmuka *Simulated Annealing Calculate*



Gambar 4.6 Antarmuka Perbandingan SA dan GA

4.4 Implementasi Uji Coba

4.4.1 Skenario Evaluasi

Pada tahap evaluasi, skenario yang digunakan sama dengan pembahasan pada subbab 3.3.5, dimana akan diterapkan beberapa aturan diantaranya banyaknya job bervariasi (10,20,30,40,50 job) pada tiap tahapnya, banyaknya mesin bervariasi (5,10,15,20,25 mesin) pada tiap tahapnya, sehingga nantinya akan terdapat 25 kombinasi simulasi. Sedangkan untuk waktu proses tiap job akan dibangkitkan nilai random antara 10 sampai 50.

Untuk parameter awal tiap algoritma akan ditentukan sebagai berikut, dimana bobot (w1 dan w2) masing-masing adalah 0,5 :

1) Algoritma Genetika
   - Chromosome number (jumlah kromosom) = 2
   - Jumlah generasi = 10000
   - Mutation Rate = 0.1

2) Algoritma Simulated Annealing
   - Temperatur Awal = 1
   - Faktor Reduksi Suhu = 0.09
   - Replikasi = 2

4.4.2 Hasil Evaluasi

Dari skenario evaluasi sebelumnya dan program dijalankan sesuai dengan aturan (masing-masing kombinasi dilakukan 15 kali percobaan), maka didapatkan hasil evaluasi dari beberapa kombinasi antara lain :

Tabel 4.1 Hasil Evaluasi Penjadwalan Menggunakan 5 Mesin

No.	Jumlah Job	Score		Prosentase Keunggulan		Waktu Proses	
		GA	SA	GA	SA	GA	SA
1	10	2	13	0,907	3,081	64381,3	49236,6
2	20	5	10	0,803	1,581	66901,2	45368,7
3	30	14	1	3,434	0,629	71316,9	43816,7
4	40	14	1	3,240	0,946	72620,9	47622,9
5	50	15	0	1,961	0	72061,5	49351,9
Total		50	25	10,345	6,237	347281,8	235396,8

Dari tabel 4.1 di atas, didapatkan nilai score dimana SA unggul terhadap GA pada job yang tidak terlalu besar (10 dan 20 job), sedangkan GA unggul terhadap SA pada job yang besar (30,40 dan 50 job) sehingga secara keseluruhan GA unggul 50 berbanding 25 terhadap SA. Untuk prosentase keunggulan algoritma dimana bobot *makespan* berbanding total *flowtime* adalah 1 : 1, didapatkan hasil bahwa algoritma *simulated annealing* dengan total rata-rata prosentase keunggulan sebesar 6,237 tidak lebih baik daripada algoritma genetika dengan total rata-rata prosentase keunggulan sebesar 10,345. Sebaliknya untuk waktu proses penjadwalan, didapatkan hasil bahwa algoritma simulated annealing dengan total rata-rata waktu proses 235396,8 ms lebih baik (lebih cepat) daripada algoritma genetika dengan total rata-rata waktu proses sebesar 347281,8 ms.

Tabel 4.2 Hasil Evaluasi Penjadwalan Menggunakan 10 Mesin

No.	Jumlah Job	Score		Prosentase Keunggulan		Waktu Proses	
		GA	SA	GA	SA	GA	SA
1	10	1	14	0,920	2,920	63541,7	49177,1

2	20	5	10	0,956	0,888	67186,5	45152,1
3	30	13	2	2,097	0,425	71340,5	43737,5
4	40	14	1	2,347	0,220	72031,3	46815,7
5	50	13	2	2,497	0,972	71269,8	48673,9
Total		46	29	8,817	5,425	345369,8	233556,3

Dari tabel 4.2 di atas, didapatkan nilai score dimana SA unggul terhadap GA pada job yang tidak terlalu besar (10 dan 20 job), sedangkan GA unggul terhadap SA pada job yang besar (30,40 dan 50 job) sehingga secara keseluruhan GA unggul 46 berbanding 29 terhadap SA. Untuk prosentase keunggulan algoritma dimana bobot *makespan* berbanding total *flowtime* adalah 1 : 1, didapatkan hasil bahwa algoritma *simulated annealing* dengan total rata-rata prosentase keunggulan sebesar 5,425 tidak lebih baik daripada algoritma genetika dengan total rata-rata prosentase keunggulan sebesar 8,817. Sebaliknya untuk waktu proses penjadwalan, didapatkan hasil bahwa algoritma simulated annealing dengan total rata-rata waktu proses 233556,3 ms lebih baik (lebih cepat) daripada algoritma genetika dengan total rata-rata waktu proses sebesar 345369,8 ms.

Tabel 4.3 Hasil Evaluasi Penjadwalan Menggunakan 15 Mesin

No.	Jumlah Job	Score		Prosentase Keunggulan		Waktu Proses	
		GA	SA	GA	SA	GA	SA
1	10	0	15	0	1,947	63910,5	49442,7
2	20	6	9	0,768	1,028	67260,5	45259,3
3	30	14	1	2,925	0,442	71796,8	44649,9
4	40	15	0	3,061	0	72295,7	46937,5
5	50	15	0	3,235	0	71469,9	48714,4
Total		50	25	9,989	3,417	346733,4	235003,8

Dari tabel 4.3 di atas, didapatkan nilai score dimana SA unggul terhadap GA pada job yang tidak terlalu besar (10 dan 20 job), sedangkan GA unggul terhadap SA pada job yang besar (30,40 dan 50 job) sehingga secara keseluruhan GA unggul 50 berbanding 25 terhadap SA. Untuk prosentase keunggulan algoritma dimana bobot *makespan* berbanding total *flowtime* adalah 1 : 1, didapatkan hasil bahwa algoritma *simulated annealing* dengan total rata-rata prosentase keunggulan sebesar 3,417 tidak lebih baik daripada algoritma genetika dengan total rata-rata prosentase keunggulan sebesar 9,989. Sebaliknya untuk waktu proses penjadwalan, didapatkan hasil bahwa algoritma simulated annealing dengan total rata-rata waktu proses 235003,8 ms lebih baik (lebih cepat) daripada algoritma genetika dengan total rata-rata waktu proses sebesar 346733,4 ms.

Tabel 4.4 Hasil Evaluasi Penjadwalan Menggunakan 20 Mesin

No.	Jumlah Job	Score		Prosentase Keunggulan		Waktu Proses	
		GA	SA	GA	SA	GA	SA
1	10	2	13	0,983	3,096	63520,9	49289,6
2	20	1	14	1,225	1,069	71100,9	45153,2
3	30	13	2	1,830	1,428	76782,3	43965,5
4	40	15	0	2,045	0	76884,3	46883,3
5	50	14	1	2,834	0,190	76900,1	48882,2
Total		45	30	8,917	5,783	365188,5	234173,8

Dari tabel 4.4 di atas, didapatkan nilai score dimana SA unggul terhadap GA pada job yang tidak terlalu besar (10 dan 20 job), sedangkan GA unggul terhadap SA pada job yang besar (30,40 dan 50 job) sehingga secara keseluruhan GA unggul 45 berbanding 30 terhadap SA. Untuk prosentase keunggulan algoritma dimana bobot *makespan* berbanding total *flowtime* adalah 1 : 1, didapatkan hasil bahwa algoritma *simulated annealing* dengan total rata-rata prosentase keunggulan sebesar 5,783 tidak lebih baik daripada algoritma genetika dengan total rata-rata prosentase keunggulan sebesar 8,917. Sebaliknya untuk waktu proses penjadwalan, didapatkan hasil bahwa algoritma simulated annealing dengan total rata-rata waktu proses 234173,8 ms lebih baik (lebih cepat) daripada algoritma genetika dengan total rata-rata waktu proses sebesar 365188,5 ms.

Tabel 4.5 Hasil Evaluasi Penjadwalan Menggunakan 25 Mesin

No.	Jumlah Job	Score		Prosentase Keunggulan		Waktu Proses	
		GA	SA	GA	SA	GA	SA
1	10	3	12	0,392	1,421	66032,5	49490,6
2	20	4	11	0,812	1,239	71276,1	45336,5
3	30	15	0	2,128	0	77032,3	44161,5
4	40	15	0	2,687	0	77449,9	47085,5
5	50	15	0	2,490	0	76892,7	49009,3
Total		52	23	8,509	2,660	368683,5	235083,4

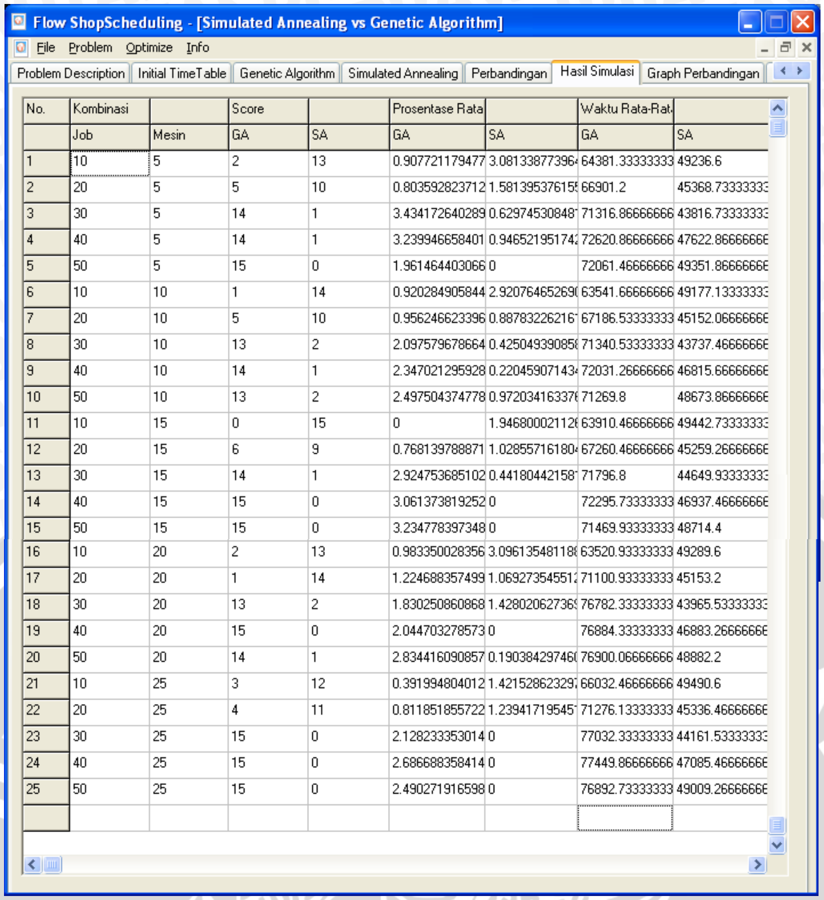
Dari tabel 4.5 di atas, didapatkan nilai score dimana SA unggul terhadap GA pada job yang tidak terlalu besar (10 dan 20 job), sedangkan GA unggul terhadap SA pada job yang besar (30,40 dan 50 job) sehingga secara keseluruhan GA unggul 52 berbanding 23 terhadap SA. Untuk prosentase keunggulan algoritma dimana bobot *makespan* berbanding total *flowtime* adalah 1 : 1, didapatkan hasil bahwa algoritma *simulated annealing* dengan total rata-rata prosentase keunggulan sebesar 2,660 tidak lebih baik daripada algoritma genetika dengan total rata-rata prosentase keunggulan sebesar 8,509. Sebaliknya untuk waktu proses penjadwalan, didapatkan hasil bahwa algoritma *simulated annealing* dengan total rata-rata waktu proses 235083,4 ms lebih baik (lebih cepat) daripada algoritma genetika dengan total rata-rata waktu proses sebesar 368683,5 ms.

Dari beberapa tabel di atas dapat dibuat tabel total rata-rata dari percobaaan semua mesin sebagai berikut :

Tabel 4.6 Hasil Evaluasi Penjadwalan Seluruh Mesin

Jumlah Mesin	Score		Prosentase Keunggulan		Waktu Proses	
	GA	SA	GA	SA	GA	SA
5	50	25	10,345	6,237	347281,8	235396,8
10	46	29	8,817	5,425	345369,8	233556,3
15	50	25	9,989	3,417	346733,4	235003,8
20	45	30	8,917	5,783	365188,5	234173,8
25	52	23	8,509	2,660	368683,5	235083,4
Total	243	132	46,577	23,522	1773257	1173214,1

Hasil di atas dapat juga dilihat pada tampilan program seperti di bawah ini.



No.	Kombinasi		Score		Prosentase Rata		Waktu Rata-Rat	
	Job	Mesin	GA	SA	GA	SA	GA	SA
1	10	5	2	13	0.907721179477	3.08133877396	64381.33333333	49236.6
2	20	5	5	10	0.803592823712	1.58139537615	66901.2	45368.7333333
3	30	5	14	1	3.434172640289	0.62974530848	71316.86666666	43816.7333333
4	40	5	14	1	3.239946658401	0.94652195174	72620.86666666	47622.8666666
5	50	5	15	0	1.961464403066	0	72061.46666666	49351.8666666
6	10	10	1	14	0.920284905844	2.92076465269	63541.66666666	49177.1333333
7	20	10	5	10	0.956246623396	0.88783226216	67186.53333333	45152.0666666
8	30	10	13	2	2.097579678664	0.42504939085	71340.53333333	43737.4666666
9	40	10	14	1	2.347021295928	0.22045907143	72031.26666666	46815.6666666
10	50	10	13	2	2.497504374778	0.97203416337	71269.8	48673.8666666
11	10	15	0	15	0	1.94680002112	63910.46666666	49442.7333333
12	20	15	6	9	0.768139788871	1.02855716180	67260.46666666	45259.2666666
13	30	15	14	1	2.924753685102	0.44180442158	71796.8	44649.9333333
14	40	15	15	0	3.061373819252	0	72295.73333333	46937.4666666
15	50	15	15	0	3.234778397348	0	71469.93333333	48714.4
16	10	20	2	13	0.983350028356	3.09613548118	63520.93333333	49289.6
17	20	20	1	14	1.224688357499	1.06927354551	71100.93333333	45153.2
18	30	20	13	2	1.830250860868	1.42802062736	76782.33333333	43965.5333333
19	40	20	15	0	2.044703278573	0	76884.33333333	46883.2666666
20	50	20	14	1	2.834416090857	0.19038429746	76900.06666666	48882.2
21	10	25	3	12	0.391994804012	1.42152862329	66032.46666666	49490.6
22	20	25	4	11	0.811851855722	1.23941719545	71276.13333333	45336.4666666
23	30	25	15	0	2.128233353014	0	77032.33333333	44161.5333333
24	40	25	15	0	2.686688358414	0	77449.86666666	47085.4666666
25	50	25	15	0	2.490271916598	0	76892.73333333	49009.2666666

Gambar 4.7 Antarmuka Hasil Simulasi SA dan GA

4.4.3 Analisa Hasil

Dari hasil evaluasi di atas, didapatkan bahwa score SA (*Simulated Annealing*) tidak selalu mengungguli GA (Genetic Algorithm) dalam setiap kombinasi. Bahkan dapat dikatakan lebih banyak GA yang mengungguli SA. Keunggulan SA hanya pada kombinasi job-job yang relatif tidak terlalu besar (10 dan 20 job), sedangkan GA selalu unggul pada job-job yang banyak (30, 40 dan 50 job) untuk setiap kombinasi jumlah mesin yang berbeda-beda. Dengan kata lain bahwa keunggulan SA atas GA atau sebaliknya hanya berpengaruh pada kombinasi jumlah job yang diinputkan (jumlah mesin yang diinputkan tidak mempengaruhi prosentase keunggulan suatu algoritma).

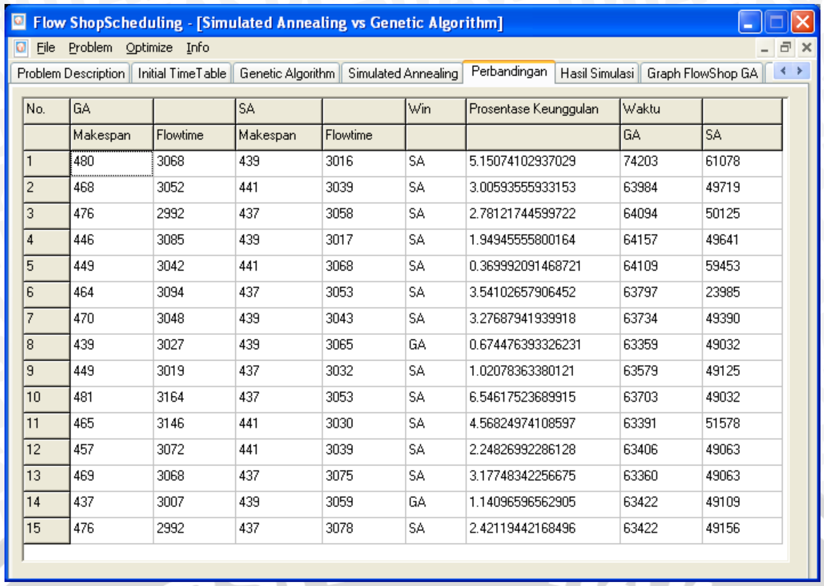
Dari segi prosentase keunggulan, GA memiliki rata-rata prosentase keunggulan yang lebih baik dibandingkan dengan rata-rata prosentase keunggulan yang dimiliki SA. Perhitungan prosentase keunggulan suatu algoritma ini diperoleh dari perbandingan nilai objektif yang dihasilkan suatu algoritma, yaitu dengan pembobotan *makespan* dan total *flowtime*, dimana ratio bobot *makespan* dibanding total *flowtime* adalah 1 : 1, yang artinya bahwa tidak ada bobot yang dititikberatkan (bobot *makespan* dan total *flowtime* seimbang).

Dari tabel 4.6, prosentase keunggulan GA atas SA adalah sebesar 23,055 % yang didapatkan dari total prosentase keunggulan GA (46,577 %) dikurangi dengan total prosentase keunggulan SA (23,522 %). Hal ini dapat terjadi karena pada saat SA unggul atas GA, nilai *makespan* SA hanya terpaut sedikit dibanding nilai *makespan* GA. Berbeda pada saat GA unggul atas SA, dimana nilai total *flowtime* GA terpaut cukup jauh dibanding nilai total *flowtime* SA.

Lain halnya dengan waktu proses rata-rata, dimana SA unggul dari GA sebesar 600042,9 ms, yang didapatkan dari jumlah total waktu proses GA (1773257 ms) dikurangi dengan total waktu proses SA (1173214,1 ms) atau dengan kata lain, waktu yang dibutuhkan SA 1,511 lebih cepat dari waktu yang dibutuhkan GA.

Pada hasil uji coba yang telah dilakukan, diperoleh bahwa GA selalu mengungguli SA. Hal ini terkait dengan skenario awal, dimana pada GA diinputkan jumlah kromosom yang kecil dan jumlah generasi yang besar yang menyebabkan GA melakukan banyak sekali proses *crossover*. Dengan crossover yang dilakukan berulang-ulang dan kromosom yang sedikit, maka pencarian solusi akhir akan lebih optimal dibandingkan pencarian yang dilakukan oleh SA dimana proses pencarian solusi akhir yang tidak terpengaruh *crossover*.

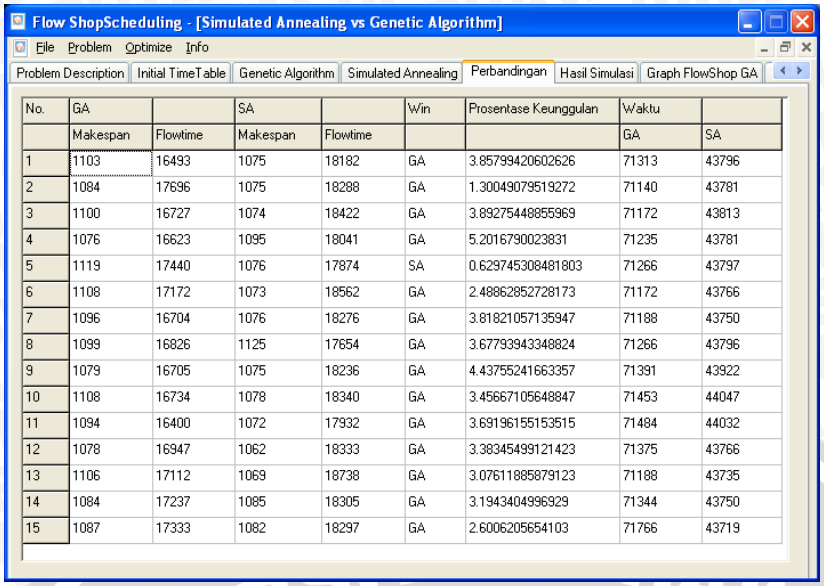
Prosentase keunggulan GA (46,577 %) yang lebih baik dibandingkan prosentase keunggulan SA (23,522 %) disebabkan oleh keunggulan kinerja SA yang hanya terjadi pada saat kombinasi job-job yang tidak terlalu besar. Lain halnya pada saat kombinasi job-job yang lebih besar, dimana GA selalu mendominasi atas SA. Hal ini dapat dilihat pada contoh berikut ini.



No.	GA		SA		Win	Prosentase Keunggulan	Waktu	
	Makespan	Flowtime	Makespan	Flowtime			GA	SA
1	480	3068	439	3016	SA	5.15074102937029	74203	61078
2	468	3052	441	3039	SA	3.00593555933153	63984	49719
3	476	2992	437	3058	SA	2.78121744599722	64094	50125
4	446	3085	439	3017	SA	1.94945555800164	64157	49641
5	449	3042	441	3068	SA	0.369992091468721	64109	59453
6	464	3094	437	3053	SA	3.54102657906452	63797	23985
7	470	3048	439	3043	SA	3.27687941939918	63734	49390
8	439	3027	439	3065	GA	0.674476393326231	63359	49032
9	449	3019	437	3032	SA	1.02078363380121	63579	49125
10	481	3164	437	3053	SA	6.54617523689915	63703	49032
11	465	3146	441	3030	SA	4.56824974108597	63391	51578
12	457	3072	441	3039	SA	2.24826992286128	63406	49063
13	469	3068	437	3075	SA	3.17748342256675	63360	49063
14	437	3007	439	3059	GA	1.14096596562905	63422	49109
15	476	2992	437	3078	SA	2.42119442168496	63422	49156

Gambar 4.8 Contoh hasil perhitungan *flow shop scheduling* pada kombinasi 5 mesin dan 10 *job*

Dari contoh gambar 4.8 (perhitungan flow shop scheduling dengan kombinasi 5 mesin dan 10 job) di atas, dapat dilihat bahwa SA hampir selalu mengungguli GA. Hal ini disebabkan nilai *makespan* dari SA yang selalu lebih baik dibandingkan nilai *makespan* yang dihasilkan GA, meskipun nilai *makespan* SA hanya unggul sedikit dibandingkan nilai *makespan* yang dihasilkan GA. Selanjutnya dapat dilihat pada gambar berikut ini.



No.	GA		SA		Win	Prosentase Keunggulan	Waktu	
	Makespan	Flowtime	Makespan	Flowtime			GA	SA
1	1103	16493	1075	18182	GA	3.85799420602626	71313	43796
2	1084	17696	1075	18288	GA	1.30049079519272	71140	43781
3	1100	16727	1074	18422	GA	3.89275448855969	71172	43813
4	1076	16623	1095	18041	GA	5.2016790023831	71235	43781
5	1119	17440	1076	17874	SA	0.629745308481803	71266	43797
6	1108	17172	1073	18562	GA	2.48862852728173	71172	43766
7	1096	16704	1076	18276	GA	3.81821057135947	71188	43750
8	1099	16826	1125	17654	GA	3.67793943348824	71266	43796
9	1079	16705	1075	18236	GA	4.43755241663357	71391	43922
10	1108	16734	1078	18340	GA	3.45667105648847	71453	44047
11	1094	16400	1072	17932	GA	3.69196155153515	71484	44032
12	1078	16947	1062	18333	GA	3.38345499121423	71375	43766
13	1106	17112	1069	18738	GA	3.07611885879123	71188	43735
14	1084	17237	1085	18305	GA	3.1943404996929	71344	43750
15	1087	17333	1082	18297	GA	2.6006205654103	71766	43719

Gambar 4.9 Contoh hasil perhitungan *flow shop scheduling* pada kombinasi 5 mesin dan 30 *job*

Berbeda dengan contoh pada gambar 4.8, dari contoh gambar 4.9 (perhitungan flow shop scheduling dengan kombinasi 5 mesin dan 30 job) di atas dapat dilihat bahwa GA hampir selalu mengungguli SA. Hal ini disebabkan nilai total *flowtime* yang dihasilkan GA selalu lebih baik dibandingkan nilai total *flowtime* yang dihasilkan SA.

Dengan ratio pembobotan *makespan* dan total *flowtime* masing-masing algoritma adalah 1 : 1 (bobot makespan dan total flowtime yang seimbang), maka dapat diketahui bahwa prosentase keunggulan GA (unggul dalam perhitungan total *flowtime*) jauh di atas prosentase keunggulan SA (unggul dalam perhitungan *makespan*). Hal ini dapat terjadi karena pada saat SA unggul atas GA, nilai *makespan* SA hanya terpaut sedikit dibanding nilai *makespan* GA. Berbeda pada saat GA unggul atas SA, dimana nilai total *flowtime* GA terpaut cukup jauh dibanding nilai total *flowtime* SA.

Kolom score merupakan nilai keunggulan suatu algoritma terhadap algoritma lain pada suatu kombinasi. Dari kolom ini dapat dilihat bahwa GA lebih baik dari SA. Total score GA adalah 243 keunggulan terhadap SA, sedangkan total score SA adalah 132 keunggulan terhadap GA. Dengan demikian GA unggul 64.8 % terhadap SA dan sebaliknya SA hanya unggul sebesar 35.2 % terhadap GA pada simulasi ini.

Dari kolom prosentase keunggulan dapat dihitung :

- Rata-rata prosentase keunggulan GA pada simulasi

$$= \frac{\sum \text{prosentase keunggulan } GA}{\sum total\ kombinasi}$$

$$= \frac{46{,}577\ \%}{25}$$

$$= 1{,}863\ \%$$

- Rata-rata prosentase keunggulan SA pada simulasi

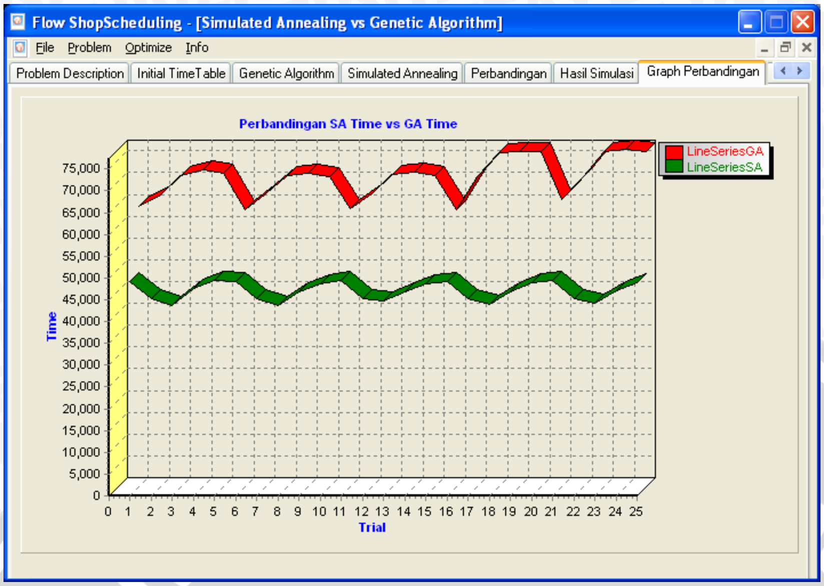
$$= \frac{\sum \text{prosentase keunggulan } SA}{\sum total\ kombinasi}$$

$$= \frac{23{,}522\ \%}{25}$$

$$= 0{,}94\ \%$$

Perhitungan di atas memang menunjukkan bahwa rata-rata prosentase keunggulan GA lebih baik dari rata-rata prosentase keunggulan SA, meskipun GA hanya unggul dalam tiap kombinasi dengan jumlah job yang besar saja dalam simulasi ini. Jadi dapat disimpulkan bahwa pada saat SA unggul, prosentase keunggulan SA hanya terpaut sedikit dari prosentase keunggulan GA, sedangkan pada saat GA unggul, prosentase keunggulan GA terpaut lebih jauh dibanding prosentase keunggulan SA.

Analisa terhadap waktu proses dapat dijelaskan bahwa rata-rata waktu proses SA lebih cepat dibanding rata-rata waktu proses GA. Prosentase keunggulan GA memang lebih baik dari SA tetapi waktu proses yang dibutuhkan juga cukup besar. Hal ini dapat dilihat pada grafik berikut ini.



Gambar 4.10 Antarmuka Grafik Perbandingan Waktu Proses Algoritma Genetika dan *Simulated Annealing*

Dari grafik di atas dapat dilihat bahwa waktu proses yang dibutuhkan SA tidak terlalu besar dan tidak terpengaruh jumlah job. Sedangkan waktu proses yang dibutuhkan GA lebih lambat pada tiap-tiap kombinasi, dimana hal tersebut dipengaruhi oleh variasi jumlah mesin dan job. Semakin meningkat jumlah job, maka waktu proses yang dibutuhkan GA untuk menyelesaikan penjadwalan semakin besar. Hal ini terlihat jelas pada kombinasi no.5, no.10, no.15, no.20 dan no.25 (yang seluruhnya menggunakan kombinasi 50 job), dimana kenaikan waktu proses sangat besar dibanding kombinasi sebelumnya yang lebih sedikit jumlah jobnya.

BAB V
KESIMPULAN DAN SARAN

5.1 Kesimpulan

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

1. Algoritma *Simulated Annealing* (SA) lebih unggul dibandingkan algoritma genetika (GA) pada saat kombinasi jumlah job yang tidak terlau besar (10 dan 20 job) dan sebaliknya, algoritma genetika (GA) lebih unggul dari *Simulated Annealing* (SA) pada saat kombinasi jumlah job yang besar (30, 40 dan 50 job) pada simulasi yang diberikan.
2. Keunggulan SA terhadap GA tidak terlalu signifikan, dikarenakan SA hanya unggul pada nilai *makespan* saja, dimana nilai *makespan* SA sedikit lebih baik dibanding nilai *makespan* GA. Sebaliknya, keunggulan GA terhadap SA cukup jauh dikarenakan nilai total *flowtime* GA yang jauh lebih baik dari nilai total *flowtime* SA.
3. Algoritma Simulated Annealing (SA) lebih unggul dari algoritma genetika (GA) pada nilai makespan yang diperoleh, tetapi rata-rata prosentase keunggulan SA masih kalah dibandingkan rata-rata prosentase keunggulan GA. Perhitungan prosentase keunggulan ini dititikberatkan pada pembobotan *makespan* dan total *flowtime* yang seimbang, dimana ratio bobot *makespan* dibanding total *flowtime* adalah 1 : 1.
4. Dalam hal waktu proses, algoritma *Simulated Annealing* (SA) lebih baik yaitu 1,511 kali lebih cepat dibandingkan algoritma genetika (GA).
5. Pada hasil uji coba yang telah dilakukan, diperoleh bahwa GA selalu mengungguli SA. Hal ini terkait dengan skenario awal, dimana pada GA diinputkan jumlah kromosom yang kecil dan jumlah generasi yang besar yang menyebabkan GA melakukan banyak sekali proses *crossover*. Dengan crossover yang dilakukan berulang-ulang dan kromosom yang sedikit, maka pencarian solusi akhir akan lebih optimal dibandingkan pencarian yang dilakukan oleh SA dimana proses pencarian solusi akhir yang tidak terpengaruh *crossover*.

5.2 Saran

Saran yang dapat diberikan setelah pengerjaan Tugas Akhir ini adalah :

1) Algoritma genetika tampak lebih baik (dari segi prosentase keunggulan) pada perhitungan kasus dengan kombinasi jumlah job yang besar, sedangkan pada kasus dengan kombinasi job yang tidak terlalu besar ternyata algoritma simulated annealing lebih unggul dari segi waktu proses dan prosentase keunggulan (walaupun SA terpaut sedikit dari GA). Sehingga kemungkinan yang dapat penulis berikan agar GA unggul pada kombinasi jumlah job yang kecil adalah dengan mengubah metode *crossover*-nya.
2) Pada uji coba dengan menginputkan jumlah kromosom yang kecil dan jumlah generasi yang besar, maka didapatkan hasil bahwa GA selalu mengungguli SA. Sehingga saran penulis agar SA bisa mengungguli GA adalah dengan mencoba menginputkan jumlah kromosom yang besar dengan jumlah generasi yang kecil.
3) Program dapat dikembangkan lagi dengan menambahkan beberapa algoritma yang lain sehingga dapat dibandingkan prosentase keunggulan maupun waktu prosesnya.

DAFTAR PUSTAKA

Baker, K. 1994. Introduction to Sequencing and Scheduling. John Wiley & Sons, Inc. New York

Cheng, C.H., W.T. Raymond. 1999. Task Scheduling by Guided Simulated Annealing. Production Planning and Control vol.10, no.6, 530-541

Dawei, L., Li, W. and Mengguang W. 1999. Genetic Algorithm for Production Lot Planning of Steel Pipe. Production Planning and Control vol.19, no.1, 54-57

French, Simon. 1982. Sequencing and Scheduling : An Introduction to the Mathematics of Job Shop. Chichester. Ellis Horwood.

Goldberg, David E. 1989. Genetic Algorithms in Search, Optimization and Machine Learning. Addison Wesley Publishing Co. Inc

Hermawanto, Denny. Algoritma Genetika dan Contoh Aplikasinya. http://www.ilmukomputer.com diakses tanggal 24 Juni 2007

Limyana, W.K. 2002. Implementasi Penjadwalan Flow Shop dengan Algoritma Genetik. Universitas Atma Jaya. Yogyakarta

Medianti, E., Vanina, T. 1999. Perbandingan Kinerja Algoritma Genetika dan Algoritma Heuristik Rajendran untuk Penjadwalan Produksi Jenis Flow Shop. Jurnal Teknik Industri vol.1 41-50

Morton, Thomas E. and Pentico, David W. 1993. Heuristic Scheduling System. Canada : John Wiley & Sons, Inc

Onwubolu, G., Mutingi, M. 1999. Genetic Algorithm for Minimizing Tardiness in Flow-Shop Scheduling. Production Planning and Control vol.10, no.5, 462-471

Ponnambalan, S.G, Jawahar, N. and Aravindan, P. 1999. A Simulated Annealing Algorithm for Job Shop Scheduling. Production Planning and Control vol.10, no.8, 767-777

Rajendran, C. 1993. Heuristic for Scheuling in Flow Shop with Multiple Objectives. European Journal of Operational Research.

Rajendran, C. and Chauduri, D. 1992. An Efficient Heuristic Approach to The Scheduling of Jobs in A Flowshops. European Journal of Operational Research no.61, 318-325