EVALUASI KINERJA *GENETIC ALGORITHM* (GA) UNTUK PENYELESAIAN PERSOALAN *JOB-SHOP SCHEDULLING PROBLEM* (JSSP)

(SKRIPSI)

Oleh

Ade Pamungkas

FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM
UNIVERSITAS LAMPUNG
BANDAR LAMPUNG
2018

ABSTRACT

PERFORMANCE EVALUATION OF A GENETIC ALGORITHM (GA) FOR SOLVING JOB SHOP SCHEDULLING PROBLEM (JSSP)

By

Ade Pamungkas

Job-Shop Scheduling Problem (JSSP) is known as one of extensively studied combinatorial optimization problems. There have been many papers reported the way for solving this problem, but to our knowledge there is no efficient solution algorithm has been found yet for solving it to optimality in polynomial time. This has led to many interests in using metaheuristic methods to solve the problem. How to develop an efficient metaheuristic method to solve job scheduling problems is still challenging but frustrating.

During the past decade, Genetic Algorithm (GA) has been one of popular metaheuristic methods for solving various difficult combinatorial problems. Many efforts have been made in order to give an efficient implementation of GA. The purpose of this research is to give a reports on performance evaluation of GA approaches in job-shop scheduling practices.

The effectiveness of GA performance is measured by solving 43 benchmark test problems consisting of 3 instances of Fisher and Thompson, (1963) and 40 instances of Lawrence, (1984). Computation results show that GA is able to solve 27 test problems with optimal and of 43 test problems tested showed that the average value of the relative error achieved only 1.14%. The results of this experiment validate that GA is effective for finding JSSP solutions.

Keywords : Genetic algorithms; Scheduling, Job-shop scheduling, Optimization problem

ABSTRAK

EVALUASI KINERJA *GENETIC ALGORITM* (GA) UNTUK PENYELESAIAN PERSOALAN *JOB SHOP SCHEDULLING PROBLEM* (JSSP)

Oleh

Ade Pamungkas

Job-Shop Scheduling Problem (JSSP) dikenal sebagai salah satu masalah optimasi kombinatorial yang banyak dipelajari. Ada beberapa literatur yang melaporkan cara untuk menyelesaikan pemasalahan ini, namun sepengetahuan kami, belum ada algoritma solusi yang efisien yang telah ditemukan untuk menyelesaikannya dengan optimalisasi pada waktu polinomial. Hal ini menyebabkan banyak ketertarikan pada penggunaan metode metaheuristik untuk mengatasi masalah tersebut. Bagaimana mengembangkan metode metaheuristik yang efisien untuk menyelesaikan persoalan JSSP masih manjadi suatu tantangan.

Selama dekade terakhir, *Genetic Algorithm* (GA) telah menjadi salah satu metode metaheuristik populer untuk menyelesaikan berbagai masalah kombinatorial yang sulit. Banyak upaya telah dilakukan untuk memberikan implementasi GA yang efisien. Tujuan penelitian ini adalah untuk memberikan laporan evaluasi kinerja dengan pendekatan GA untuk menyelesaikan JSSP.

Efektivitas kinerja GA diukur dengan menyelesaikan 43 *benchmark test problem* yang terdiri dari 3 *instance* dari Fisher dan Thompson, (1963) dan 40 *instance* dari Lawrence, (1984). Hasil komputasi menunjukan bahwa GA mampu menyelesaikan 27 *test problem* dengan optimal dan dari 43 test problem yang diujikan menunjukan bahwa nilai rata rata *relatif error* yang dicapai hanya 1,14%. Hasil eksperimen ini memvalidasi bahwa GA efektif untuk menemukan solusi JSSP.

Kata Kunci: *Genetic algorithms; Scheduling, Job-shop scheduling; Optimization problem*

EVALUASI KINERJA *GENETIC ALGORITHM* (GA) UNTUK PENYELESAIAN PERSOALAN *JOB-SHOP SCHEDULLING PROBLEM* (JSSP)

Oleh

Ade Pamungkas

Skripsi

Sebagai Salah Satu Syarat untuk Memperoleh Gelar
SARJANA KOMPUTER

Pada

Program Sudi Ilmu Komputer
Fakultas Matematika dan Ilmu Pengetahuan Alam

JURUSAN ILMU KOMPUTER
FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM
UNIVERSITAS LAMPUNG
BANDAR LAMPUNG
2018

Judul Skripsi	:	**EVALUASI KINERJA *GENETIC ALGORITHM* (GA) UNTUK PENYELESAIAN PERSOALAN *JOB-SHOP SCHEDULLING PROBLEM* (JSSP)**
Nama Mahasiswa	:	Ade Pamungkas
No. Pokok Mahasiswa	:	1117032001
Jurusan	:	Ilmu Komputer
Fakultas	:	Matematika dan Ilmu Pengetahuan Alam

MENYETUJUI

1. Komisi Pembimbing

Dr. Eng. Admi Syarif	**Dr. Ir. Kurnia Muludi, M.S.Sc.**
NIP 19670103 199203 1 003	NIP 19640616 198902 1 001

2. Mengetahui
Ketua Jurusan Ilmu Komputer
FMIPA Universitas Lampung

Dr. Ir. Kurnia Muludi, M.S.Sc.
NIP 19640616 198902 1 001

MENGESAHKAN

1. Tim Penguji

   Ketua : **Dr. Eng. Admi Syarif** .....

   Sekretaris : **Dr. Ir. Kurnia Muludi, M.S.Sc.** .....

   Penguji
   Bukan Pembimbing : **Aristoteles, S.Si., M.Si.** .....

2. Dekan Fakultas Matematika dan Ilmu Pengetahuan Alam

Prof. Warsito, S.Si. D.E.A., Ph.D.
NIP. 19710212 199512 1 001

Tanggal Lulus Ujian Skripsi : **23 Januari 2018**

PERNYATAAN

Saya yang bertanda tangan di bawah ini, menyatakan bahwa skripsi saya yang berjudul "**EVALUASI KINERJA *GENETIC ALGORITHM* (GA) UNTUK PENYELESAIAN PERSOALAN *JOB-SHOP SCHEDULLING PROBLEM* (JSSP)**" ini merupakan hasil karya sendiri dan bukan hasil karya orang lain. Semua hasil tulisan yang tertuang dalam skripsi ini telah mengikuti kaidah penulisan karya ilmiah Universitas Lampung. Apabila di kemudian hari terbukti bahwa skripsi ini merupakan hasil penjiplakan atau dibuat oleh orang lain, maka saya bersedia menerima sanksi sesuai hukum yang berlaku.

Bandar Lampung, 9 Februari 2018

Ade Pamungkas
NPM. 1117032001

RIWAYAT HIDUP

Penulis dilahirkan pada tanggal 25 April 1991 di Desa Sidodadi, Kecamatan Bangun rejo, Kabupaten Lampung Tengah. Penulis adalah putra keenam dari tujuh bersaudara, dari pasangan Bapak Khojin Pramono dan Ibu Warsini. Penulis menempuh pendidikan sekolah dasar di SD Negeri 2 Sidodadi pada tahun 1998 dan selesai pada tahun 2004, kemudian melanjutkan pendidikan Sekolah Menengah Pertama di SMP Negeri 1 Kalirejo, lulus pada tahun 2007 dan Sekolah Menengah Atas di SMA Negeri 1 Kalirejo, lulus pada tahun 2010.

Pada tahun 2011 penulis terdaftar sebagai mahasiswa Program Studi Ilmu Komputer, Jurusan Ilmu Komputer, Fakultas Matematika dan Ilmu pengetahuan Alam, Universitas Lampung melalui jalur SMPTN (Seleksi Nasional Masuk Perguruan Tinggi Negeri), Selama menjadi mahasiswa, penulis aktif mengikuti kegiatan korganisasisian, diantaranya adalah. Anggota Muda Himpunan Mahasiswa Matematika (HIMATIKA) Universitas Lampung periode 2011-2012. Anggota Komunitas Mahasiswa Ilmu Komputer (KoMIK) Universitas Lampung periode 2011-2012. Anggota Himpunan Mahasiswa Ilmu Komputer (HIMAKOM) Universitas Lampung periode 2012-2013. Komandan Korps Muda BEM (KMB 7) Universitas Lampung periode 2011-2012. Kepala Bidang Kehumasan Rohani

Islam (ROIS) FMIPA Universitas Lampung pada periode 2013-2014. Penulis juga aktif di berbagai seminar seperti "*The International Conference on Science, Technology, and Interdisciplinary Research* (IC-STAR) yang diadakan oleh Universitas Lampung, "*The USR International Seminar on Food Security* (UISFS)" yang diadakan oleh Universitas Lampung bekerja sama dengan SEAMEO-SEARCA dan RSFA (*Regional Searca Fellow Association*), dan juga "*International Wildlife Symposium* 2016" yang diadakan oleh Universitas Lampung bekerja sama dengan WWF.

Pada Bulan Juli sampai dengan Agustus 2014, penulis melaksanakan Kerja Praktik (KP) di UPT Pusat komputer (Puskom) Universitas Lampung dengan topik penelitian "Pengembangan Sistem Informasi Absensi Berbasis Web di Universitas Lampung". Pada bulan Januari sampai dengan Maret 2015, penulis melaksanakan Kuliah Kerja Nyata (KKN) di Desa Sumber Makmur, Kecamatan Banjar Margo, Kabupaten Tulang Bawang, Provinsi Lampung.

MOTTO

" Kunci rahasia kesuksesan besar ada bersama 'Bismillah' "
(Ade Pamungkas)

"Tolonglah orang yang datang kepadamu sebisa mungkin, karena bisa jadi dilain waktu dia tidak datang kepadamu lagi maka itu merupakan kerugian besar bagimu karena tidak bisa menolong sesama "
(Dr. Eng. Admi Syarif)

"Takutlah kamu dengan firasat seorang mukmin. Sebab ia memandang sesuatu dengan cahaya Illahi." (Al hadist)

"Tarekat kita adalah persahabatan (kebersamaan), dan kebaikan berada dalam kebersamaan"
(Hadrat Khwaja Syekh Bahauddin Naqsyabandi q.s.)

PERSEMBAHAN

Kupersembahkan karya yang dibuat dengan penuh semangat, pengorbanan, kerja keras, doa dan cinta ini kepada:

"Ibu Warsini dan Bapak Khojin Pramono"

Ayah dan Ibu Terhebat di dunia

KATA PENGANTAR

Puji syukur penulis panjatkan kehadirat Allah SWT, yang telah memberikan kesehatan dan karunia-Nya sehingga penulis dapat menyelesaikan skripsi yang berjudul **"EVALUASI KINERJA *GENETIC ALGORITHM* (GA) UNTUK PENYELESAIAN PERSOALAN *JOB-SHOP SCHEDULLING PROBLEM* (JSSP)".** Tujuan penulisan skripsi ini adalah sebagai salah satu persyaratan untuk mendapatkan gelar S1 dan melatih mahasiswa untuk berpikir cerdas, kreatif dan holistik dalam menulis karya ilmiah. Penulis menyadari masih banyak kekurangan dalam skripsi ini.

Penulis mengharapkan kritik dan saran yang membangun. Akhir kata, semoga skripsi ini dapat bermanfaat bagi semua. Aamin yarabbal alamiin.

Bandar Lampung, 9 Februari 2018

Penulis,

Ade Pamungkas

SANWACANA

Assalamualaikum Wr.Wb

Syukur Alhamdulillah, puji syukur penulis ucapkan kehadirat Allah SWT, karena atas rahmat dan hidayah-Nya penulisan karya tulis ilmiah ini dapat diselesaikan. Shalawat beserta salam tercurah kepada Abu Qasim, Nabi besar Muhammad SAW sang kekasih tuhan semesta alam.

Skripsi yang berjudul "**Evaluasi Kinerja Genetic Algorithm (GA) untuk Penyelesaian Persoalan *Job-shop Schedulling Problem* (JSSP)**" adalah salah satu syarat untuk memperoleh gelar sarjana Ilmu Komputer di Universitas Lampung.

Dalam kesempatan ini penulis mengucapkan terimakasih kepada:

1. Ayah dan Ibuku tercinta, Ibu Warsini dan Bapak Khojin pramono, yang selalu mendoakan dan bekerja keras untuk mendukung segala cita-citaku. Terimkasih bu, pak atas segala curahan kasih sayang ibu dan bapak, tak mungkin aku bisa membalasnya. Semoga Allah SWT selalu memberikan rahmat dan kasih sayang-Nya kepada ayah dan ibu ku.

2. Bapak Dr. Eng. Admi Syarif, sebagai pembimbing utama yang selalu memberikan bimbingan kepada penulis dan memberikan dorongan semangat sehingga penulisan ini dapat diselesaikan.
3. Bapak Dr. Ir. Kurnia Muludi, M.S.Sc., sebagai pembimbing kedua dan selaku Ketua Jurusan Ilmu Komputer yang telah memberikan saran dan membimbing penulis dalam penulisan skripsi ini.
4. Bapak Aristoteles, S.Si., M.Si., sebagai penguji yang telah memberikan masukan, kritikan dan saran dalam perbaikan skripsi ini.
5. Ibu Anie Rose Irawati, S.T., M.Cs., dan Bapak Warsono, Ph.D., selaku pembimbing akademik selama penulis menjadi mahasiswa Ilmu Komputer Universitas Lampung.
6. Bapak Didik Kurniawan, M.T., selaku sekretaris Jurusan Ilmu Komputer Fakultas Matematika dan Ilmu Pengetahuan Alam.
7. Bapak Prof. Dr. Warsito, S.Si., DEA., selaku dekan Fakultas Matematika dan Ilmu Pengetahuan Alam.
8. Seluruh dosen dan karyawan Jurusan Ilmu Komputer Fakultas Matematika dan Ilmu Pengetahuan Alam.
9. Keluarga besarku tercinta, kakek dan nenekku, Mbah Siti Mahmudah, Mbah Bainah, Mbah Suradi, Mbah Martowijoyo (alm), Paman dan bibiku, kakak dan adikku tersayang. Kang Wahid Priyanto, Yayu Iswati, Yayu Maryati, Yayu Nur, Kang Heru, Gendo Wati,.S.Pd, Mba Rika, Mas Herman, Mas Ikin. Ponakanku Robi, Aqila, Rafi, Athfan, Dani, Daffa. Sepupuku Ristia fatimah, Nina Nurmala, Yudha, Mas Ari, Tripujiana, Mas Heri, Mba Yuli, Rohmah,

Nani, Rohim, Lia,Via, Ifan, Lulu, Azril, Sisri, Fitri, Aji, Nurul, semuanya yang selalu bisa berbagi kebahagiaan.

10. Guru Spiritualku yang telah membina ruhaniku, Kang Ustadz. Ujang Bukhori, Ustadz. Syamsul Rizal, M.Si., Ustadz. Trimulyono, Lc. M.H., Ka Agus Solihin, S.Pi.
11. Keluarga besar ilmu komputer angkatan'11, Adi Sariyadi, Adi Wijaya, Basir, Budi, Gamma, Tryo, Rudra, Yunita, Novita, Clara, Ika, Maya, Dea, Harry, Harisa, Aqila, Panji, Dimas, Oky, Ardika, Pandya, Aldona, Tika, Ardye, Ardi, Risqi, Azaricho, Dona, Riska, Ika, Dimas, Rahmat, Putri, Aldona, Amir, Galih, Bobby, Sigit, Albion, Ghozy, Cinthya, Jonhar, Bayu Briandita, Fitriana, Pradana dan rahimahullah Ismail Indra pratama, semoga allah melapangkan kuburnya.
12. Teman berjuang yang selalu berbagi cita dan semangat, Imam Mukhlasin, Fery Ferdianto, Ahmad Said Abdurahman, Yulianto, Andi Fajar Syamsudin, Bib Sammy Rizki Taufik , Bang Ali Akbar Hasibuan, Mas Ari Susanto, Muhammad Sobran Jamil, Akh Wahyudi Pangestu, Kang Irkham Bariklana. Dr. Guntur Sulisty, Akh Nurudin , Angga Riyadi, Aan Pratama, Beny Tribiyono, Aziez Nur Dwiyansyah, Oka Amsal, Isnaeni Rahmadi.
13. Kakanda yang selalu menyayangiku, Ka Adi Mendoza, Ka Hafidz, Ka Tri Sujarwo Songha, Ka Umam Fahmi.
14. Adikku satu perjuangan yang sangat saya banggakan Andi Zulkarnaen, Aan, Kyay Iqbal Yuliansyah, Akh Allaudin Alayubi, Ubay, Arif Furqon, Agum, Wahyudi,Dimas rahmat Adi, Hamid, Suyit, Aiman, Zainuri, Mujahid, Ganjar, Arif Aulia, Rosyad, Afif Luthfi, Pranoto, Efrizal, Rifal Kassa Dinar, Anis,

Midin, Erva Alhusna, Dini Khansa, Eria Ayu Aningtyas, Uli Saragih, Anisa Nurfadila, Jean, Ruwai, Fentri, Taqiya, Ulafatun nurun, Aulia, Lina, Hilya, Uut, Melita, Winni Rahmawati.

15. Teman-teman Ilmu komputer angkatan 2010, 2012, 2013, Rois FMIPA, HIMAKOM, KMB 7 Terimakasih atas pertemanan yang sangat berkesan.
16. Pak Igo dan seluruh penghuni rumah kita, Ka Kholil, Anwar, Bayu, Ahmad Nur, Heru yang telah berbagi keceriaan bersama.
17. Semua pihak yang tidak tersebutkan yang telah membantu penyelesaian Skripsi ini, penulis ucapkan juga terima kasih atas segala bantuan dan sarannya.

Akhir kata hanya Allah SWT pemilik kesempurnaan dan penulis hanya seorang faqir ilmu, skripsi ini masih jauh dari sempurna untuk itu diharapkan saran dan kritik yang membangun. Semoga skripsi ini dapat bermanfaat bagi perkembengan ilmu pengetahuan dan bagi semua yang membacanya. Amin.

Wasalamualaikum Wr.Wb

Bandar Lampung, 9 Februari 2018

Penulis

Ade Pamungkas

DAFTAR ISI

DAFTAR GAMBAR

Gambar **Halaman**

DAFTAR TABEL

Tabel **Halaman**

I. PENDAHULUAN

1.1 Latar Belakang

Penjadwalan memiliki peran sebagai teknologi inti dalam sistem produksi. Semua industri membutuhkan penjadwalan yang tepat untuk pengaturan pengalokasian sumber daya agar sistem produksi berjalan dengan cepat dan tepat sehingga mendaptkan hasil produksi yang optimal. Pinedo, (2008) mengemukakan bahwa Penjadwalan adalah proses pengambilan keputusan yang digunakan secara teratur di banyak industri manufaktur dan jasa. Ini berkaitan dengan alokasi sumber daya untuk melaksanakan tugas selama periode waktu tertentu dan tujuannya adalah untuk mengoptimalkan satu atau lebih tujuan. Fera *et al,* (2015) juga menjelaskan bahwa penjadwalan pada dasarnya adalah rencana pelaksanaan jangka pendek dari model perencanaan produksi. Suatu penjadwalan adalah persoalan penugasan yang menjelaskan sampai rinci aktivitas yang harus dilakukan dan bagaimana sumber daya pabrik harus digunakan untuk memenuhi rencana. Pengertian penjadwalan tersebut dapat dijabarkan bahwa panjadwalan merupakan konsep yang berisikan prinsip, model dan teknik untuk proses pengambilan keputusan.

Saat ini penelitian tentang penjadwalan menjadi salah satu topik populer dalam penelitian di bidang riset operasi. Penelitian menekankan pada persoalan penjadwalan mesin (*machine shchedulling problem)* dengan *jobs*

merepresentasikan aktivitas dan mesin merepresentasikan sumber daya, setiap mesin bisa memproses paling banyak satu pekerjaan pada satu waktu. (Yamada & Nakano, 1997). Masalah penjadwalan mesin yang sering ditemui dalam proses produksi yaitu *Flow-shop Secheduling Problem* (FSSP) dan *Job-Shop Scheduling Problem* (JSSP).

JSSP adalah persoalan penjadwalan satu set pekerjaan pada satu set mesin, tergantung pada batasan yang ada. Setiap mesin dapat menangani paling banyak satu pekerjaan pada satu waktu serta setiap pekerjaan memiliki urutan pemrosesan tertentu melalui mesin, tujuannya adalah untuk menjadwalkan pekerjaan sehingga meminimalkan maksimal waktu penyelesaiannya (*makespan*). JSSP telah menjadi topik riset yang menarik dalam kurun waktu yang cukup lama dan menjadi suatu tantangan komputasi di bidang riset operasi, karena JSSP termasuk jenis persoalan *NP-Hard problem* dan merupakan salah satu masalah yang paling sulit dalam domain optimasi kombinatorial (Garey *et al*, 1976; Lenstra & Rinnooy Kan, 1979). *NP hard problem* adalah akronim dari *Non-deterministic Polynomial hard problem* yang berarti ketika ukuran persoalan naik maka waktu untuk menemukan solusi optimal akan naik secara eksponensial.

Ada dua pendekatan dasar yang digunakan untuk menyelesaikan JSSP, yaitu pendekatan metode eksak dan pendekatan metode aproksimasi. Metode eksak mampu memecahkan masalah berskala kecil dengan baik akan tetapi semua metode eksak tidak bisa memecahkan masalah berskala besar dalam waktu yang layak (Akram , *et al*., 2016). Beberapa metode eksak yang pernah diaplikasikan untuk penyelesaian *job* shop adalah metode enumerasi (Lageweg, *et al*., 1977), *integer*

programming (Özgüven, *et al*.,2012 ; Catanzaro, *et al*.,2015), *Lagrangian relaxation* (Baptiste, *et al.*, 2006), *dynamic programming* (Gromicho, *et al*., 2012), dan metode *branch and bound* (Brucker, *et al.*, 2012).

Penerapan metode aproksimasi (*metaheuristik*) mampu menyelesaikan kasus penjadwalan skala besar secara efisien (Fera, *et al.*, 2015). Beberapa metode aproksimasi yang telah diterapkan untuk penyelesaian JSSP yaitu *Genetic Algorithm* (Hou, *et al.*, 2011), *Tabu Search* (TS) (Nowicki & Smutnicki, 1996), *Simulated Annealing*(SA) (Akram , *et al*., 2016), *Electromagnetism-like Algorithm* (Roshanaei , *et al*., 2010), *Ant Colony Optimization* (ACO) (Flórez, *et al*., 2013), *Particle Swarm Optimization* (PSO) (Pongchairerks, 2009) dan *Artificial Immune System* (AIS) (Muhamad, *et al*., 2015). Beberapa penelitian telah berhasil menggabungkan beberapa metode aprokimasi untuk mendapatkan hasil optimisasi yang lebih baik, diantaranya yaitu penggabungan metode GA dengan *Local Search* (LS)(Gonçalves, *et al.*, 2005; El-Desoky , *et al*., 2016), Penggabungan metode *Harmony Search* (HS) dengan *Bacterial Foraging* (BFO) (Shivakumar & Amudha, 2012), Penggabungan metode *Harmony Search* (HS) dengan *Local Search* (LS) (Piroozfard, *et al.*, 2015).

Ada beberapa fungsi tujuan untuk persoalan JSSP tetapi yang sering dijumpai pada banyak literatur yaitu meminimasi *makespan*. Beberapa penelitian yang sudah dilakukan untuk meminimumkan *makespan* di antaranya dilakukan oleh Nowicki dan Smutnicki (1996), Yamada dan Nakano (1997), Gonçalves *et al* (2005), Mesghouni dan Hammadi (2004), Pongchairerks (2009), Sha dan Lin (2009), Shivakumar B L dan Amudha (2012), Flórez, *et al* (2013), Muhamad *et al* (2015),

Piroozfard, *et al*. (2015), El-Desoky *et al* (2016), Akram , *et al.*, (2016). Penelitian dengan fungsi tujuan yang lain seperti meminimumkan rata rata *flow time* (rata rata lamanya waktu yang dihabiskan seluruh *job* pada lantai produksi) telah dilakukan juga oleh Sotskov dan Shakhlevich (1995), Mehmood *et al* (2013).

Genetic algorithm (GA) adalah program komputer yang meniru proses evolusi biologis untuk menyelesaikan masalah dan memodelkan sistem evolusioner (Mitchell, 1995). GA adalah teknik pencarian yang kuat berdasarkan evolusi biologis alami yang digunakan untuk menemukan solusi optimal atau mendekati solusi optimal (El-Desoky , *et al.*, 2016). GA merupakan metode adaptif yang dapat digunakan untuk penyelesaian persoalan pencarian dan optimasi. GA memiliki keunggulan sebagai teknik pencarian yang telah terbukti *robust* (tangguh), dan dapat menangani dengan sukses berbagai area masalah yang termasuk sulit diselesaikan oleh metode lain (Beasley, *et al.*, 1993). Selain memberikan hasil yang cukup baik atau bisa diterima, GA juga mampu memberikan suatu solusi dalam waktu yang singkat. GA adalah salah satu metode alternatif yang bisa digunakan untuk penyelesaian persoalan JSSP berskala besar.

Pada saat ini GA dikenal telah banyak digunakan untuk menyelesaikan persoalan komputasi yang komplek. GA telah banyak diaplikasikan ke berbagai model dan persoalan sains dan teknik seperti persoalan optimasi (*Optimization*), pemrograman otomatis (*Automatic Programming*), pembelajaran mesin (*Machine Learning*), pemodelan ekonomi *(Economic Model*), pemodelan sistem imun (*Immune System Model*) dan Pemodelan ekologis (*Ecological Models)* (Mitchell, 1995) .

GA memiliki kelebihan dan keterbatasan yang selanjutnya bisa menjadi bahan pertimbangan untuk diterapkan ke berbagai jenis persoalan yang berbeda. Kelebihan GA yaitu memiliki ruang solusi yang luas sehingga memudahkan untuk pencarian nilai optimum global dan tidak mudah terjebak pada nilai optimum lokal. GA Hanya menerapkan evaluasi fungsi pada persoalan yang akan diselesaikan. GA sangat cocok untuk diterapakan pada persoalan yang memiliki kompleksitas cukup rumit dan memiliki fungsi tujuan yang banyak. GA dapat dengan mudah dimodifikasi untuk diterapakan pada persoalan yang berbeda dan sangat handal untuk mengevaluasi fungsi tujuan yang kompleks. GA dapat bekerja dengan sangat baik untuk persoalan optimasi berskala besar dan dapat diterapkan secara luas pada berbagai jenis persoalan optimasi. Meskipun GA adalah metode yang handal GA juga memiliki beberapa keterbatasan yaitu ditahap awal sulit untuk mengidentifikasi fungsi kelayakan (*fitness*) serta sulit dalam mendefinisikan representasi masalah dan tidak baik dalam mendefinisikan nilai optimum lokal. Penyelesaian JSSP memungkinkan banyak sekali solusi yang didapat oleh karena itu sangat tepat untuk diselesaikan dengan GA untuk mendapatkan solusi yang paling optimal (Sivanandam & Deepa, 2008).

1.2 Rumusan Masalah

Berdasarkan latar belakang permasalahan di atas, maka rumusan masalah pada penelitian ini adalah “ Bagaimana evaluasi hasil kinerja GA untuk penyelesaian persoalan *Job-shop Scheduling Problem* (JSSP) ?”.

1.3 Tujuan

Tujuan dari penelitian ini adalah

1. Mengimplementasikan GA pada persoalan JSSP
2. Mengevaluasi kinerja GA pada JSSP
3. Membandingkan kinerja GA pada JSSP dengan metode lain.

1.4 Manfaat

Hasil penelitiaan ini diharapkan dapat bermanfaat untuk :

1. Membantu memenuhi kebutuhan industri terhadap kebutuhan penjadwalan yang efektif
2. Memberikan formulasi metode penyelesaian JSSP yang efektif dan efisien
3. Menambah pustaka untuk penelitian lebih lanjut pada topik terkait penjadwalan dan atau GA

II. TINJAUAN PUSTAKA

2.1 Penjadwalan (*scheduling*)

Salah satu pokok persoalan terpenting di dalam proses perencanaan dan operasi di sistem manufaktur yaitu proses penjadwalan (Zhang, *et al*., 2008). Penjadwalan (*scheduling*) adalah proses pengalokasian sumber daya atau mesin yang ada untuk menjalankan sekumpulan pekerjaan dalam jangka waktu tertentu (Baker, 1974). Menurut Morton dan Pentico (1993) sistem penjadwalan secara dinamis adalah proses pengambilan keputusan tentang penyesuaian aktivitas dan sumber daya untuk menyelesaikan pekerjaan atau proyek secara tepat waktu dan sekaligus memaksimalkan hasil yang bermutu tinggi dan meminimalkan biaya operasional langsung. Jenis keputusan penjadwalan dasar yang dibuat meliputi :

1. Pengurutan pekerjaan (*sequencing*)
2. Pengaturan waktu (*timing*)
3. Perutean atau pengurutan operasi untuk suatu pekerjaan *(routing*)

Dalam terminologi penjadwalan terdapat perbedaan pengertian antara pengurutan (*sequencing*) dan penjadwalan (*scheduling*). penjadwalan mengacu pada pengalokasian sumber daya untuk menyelesaikan pekerjaan, penjadwalan merupakan proses penugasan kapan pekerjaan harus dimulai dan diselesaikan dengan memenuhi *constrains* yang ada, sedangkan pengurutan (*sequencing*)

merupakan proses pengaturan urutan atas pekerjaan-pekerjaan yang harus diselesaikan tersebut. Penjadwalan banyak sekali dijumpai dalam kehidupan sehari hari. Berikut adalah beberapa contoh situasi yang membutuhkan teknik pengurutan dan penjadwalan (Alharkan, 2005)

1. Bagian produksi yang menunggu waktu proses di suatu pabrik
2. Pesawat terbang yang menunggu izin mendarat di bandara
3. Program komputer yang berjalan di pusat komputasi
4. Penjadwalan kelas di sekolah
5. Seorang pasien yang menunggu di ruang dokter
6. Kapal berlabuh di pelabuhan
7. Pekerjaan sehari hari di rumah

Menurut Baker, (1974) ada tiga jenis tujuan pengambilan keputusan secara umum dalam masalah pengurutan dan penjadwalan:

1. Efisiensi pemanfaatan sumber daya.

   Aktivitas pejadwalan harus memastikan pemanfaatan tenaga kerja, peralatan dan ruang secara efesien.
2. Respon cepat terhadap permintaan (*demands*)

   Penjadwalan harus memungkinkan pekerjaan diproses dengan kecepatan tinggi sehingga mengurangi ketersediaan barang setengah jadi.
3. Memenuhi kesesuaian dengan *deadline*.

   Setiap pekerjaan mempunyai batas waktu (*due date*) penyelesaian, jika pekerjaan tersebut diselesaikan melewati batas waktu yang ditentukan maka pekerjaan tersebut dinyatakan terlambat. Penjadwalan harus memastikan

bahwa batas waktu penyelesaian suatu pekerjaan terpenuhi, sehingga meminimalisir keterlambatan penyelesaian suatu pekerjaan.

2.1.1 Level Pengurutan dan Penjadwalan

Pengurutan dan penjadwalan terlibat dalam perencanaan dan pengendalian proses pengambilan keputusan industri manufaktur dan jasa dalam beberapa tahap. Pengurutan dan penjadwalan ada pada beberapa tingkat proses pengambilan keputusan. Tingkatan ini dikelompokan sebagai berikut ini (Morton & Pentico, 1993):

1. Perencanaan jangka panjang *(long-term planning*)

   Memiliki rentang 2 sampai 5 tahun. Beberapa contohnya adalah: *plant layout, plant design,* dan *plant expansion.*

2. Perencanaan jangka menengah (*middle-term planning*)

   dilakukan dalam kurun waktu 1 sampai 2 tahun seperti *production smoothing*.

3. Perencanaan jangka pendek (*short-term planning*)

   Dilakukan setiap 3 atau 6 bulan. Contohnya meliputi: *requirements plan, shop bidding,* dan *due date setting.*

4. Penjadwalan prediktif (*predictive scheduling*)

   Dilakukan dalam rentang 2 sampai 6 minggu misalnya *Job shop routing*.

5. Penjadwalan reaktif (*reactive scheduling or control*)

   Dilakukan setiap hari atau setiap tiga hari. Contohnya *hot jobs, down machines,* dan *late material*

2.1.2 Lingkungan Penjadwalan

Menurut Conway *et al.*, (1967), lingkungan Pengurutan dan penjadwalan diklasifikasikan menurut empat jenis informasi:

1. Pekerjaan dan operasi yang akan diproses.
2. Jumlah dan jenis mesin.yang meliputi beberapa shop
3. Disiplin yang membatasi cara penugasan pekerjaan dapat dilakukan.
4. kriteria dimana jadwal akan dievaluasi.

Beberapa lingkungan pengurutan dan penjadwalan adalah sebagai berikut (Alharkan, 2005) :

1. *Single machine shop*: satu mesin dan n *job* untuk diproses.
2. *Flow shop*: Ada mesin sejumlah m dalam seri dan *job* dapat diproses dengan salah satu cara berikut:
    a. Permutational: *Job* diproses oleh serangkaian mesin m dengan urutan yang sama persis
    b. Non-Permutasional: *Job* diproses oleh serangkaian mesin m yang tidak berurutan sama.
3. *Job shop:* setiap *job* memiliki pola alir dan subset dari *job* ini dapat mengunjungi setiap mesin dua kali atau lebih.
4. *Assembly job shop*: *job shop* dengan *job* yang memiliki setidaknya dua komponen item dan setidaknya satu kumpulan operasi.
5. *Hybrid job shop*: prioritas pengurutan operasi beberapa *job* adalah sama.
6. *Hybrid assembly job shop*: menggabungkan fitur baik dari *Assembly job shop* dan *Hybrid job shop*.

7. *Open shop*:

   Ada sejumlah *m* mesin dan tidak ada batasan dalam perutean masing-masing pekerjaan melalui mesin. Dengan kata lain, tidak ada pola aliran spesifik untuk *job* manapun.

8. *Closed shop*:

   Semua pesanan produksi dihasilkan sebagai hasil keputusan penambahan persediaan dengan kata lain, produksi tidak terpengaruh oleh pesanan pelanggan.

2.1.3 Kategori Masalah Penjadwalan

Model penjadwalan dapat di kategorikan berdasarkan beberapa keadaan. Berikut kategori masalah pengurutan dan penjadwalan (Baker & Trietsch, 2009):

1. Berdasarkan mesin yang digunakan
   a. *Single machine*

      Pengurutan dan penjadwalan pada mesin tunggal.

   b. *Multiple machine*

      Pengurutan dan penjadwalan pada mesin majemuk

2. Berdasarkan Pola kedatangan pekerjaan
   a. Dinamis

      Himpunan pekerjaan berubah dari waktu ke waktu dan pekerjaan tiba pada waktu yang berbeda.

   b. Statis

Pekerjaan datang secara bersamaan pada waktu nol. himpunan pekerjaan yang tersedia untuk penjadwalan tidak berubah dari waktu ke waktu, dan ini sudah tersedia sebelumnya.

3. Berdasarkan sifat informasi yang diterima
   a. Deterministik

      sifat dan informasi yang diperoleh bersifat pasti atau ketika kondisi diasumsikan diketahui dengan pasti seperti *due date* dari suatu *job*, pengurutan, waktu pemrosesan,ketersediaan mesin.

   b. Stokastik

      sifat dan informasi yang diperoleh tidak pasti. setidaknya salah satu unsur masalah termasuk faktor stokastik.

2.1.4 Notasi dalam Penjadwalan

Dalam semua masalah penjadwalan, jumlah *job*s dan jumlah mesin diasumsikan tidak berhingga. Jumlah *jobs* dinotasikan dengan n dan jumlah mesin dinotasikan dengan m. Biasanya subskript *j* mengacu pada *job* dan subskript i mengacu pada mesin, jika pekerjaan membutuhkan sejumlah langkah pemrosesan atau operasi, maka pasangan (i, j) mengacu pada langkah atau operasi pemrosesan *job j* pada mesin i. Notasi yang biasa digunakan dalam penjadwalan produksi jika terdapat sejumlah n *job* yang harus diproses pada sejumlah m mesin dapat dijelaskan seperti berikut (Baker & Trietsch, 2009):

1. *Release date* atau *Ready time* (r_j)

   waktu yang menunjukkan saat *job* siap untuk dikerjakan

2. *Flowtime* (F_j)

   Waktu yang diperlukan oleh suatu *job* j dikerjakan dalam sistem sampai *job* tersebut selesai. $F_j = C_j - r_j$

3. *Processing time* (P_{ij})

   Merupakan waktu pemrosesan *job* j pada mesin i. Subskrip i dihilangkan jika waktu pemrosesan *job* j tidak bergantung pada mesin atau jika *job* j hanya diproses pada satu mesin saja.

4. *Process time* (t_{ij})

   Process time yaitu waktu yang diperlukan untuk menyelesaikan suatu operasi atau proses ke- i dari *job* ke- j. Waktu proses ini telah mencakup waktu untuk persiapan dan pengaturan proses,

5. *Lateness* (L_j)

   Penyimpangan waktu penyelesaian suatu *job* ke-j hingga saat *due date* .

   $L_j = C_j - d_j \leq 0$

   < 0, jika penyelesaian memenuhi batas akhir

   > 0, jika penyelesaian melewati batas akhir

6. *Tardiness* (T_j)

   besarnya keterlambatan dari *job* j hingga saat *due date* . *Tardiness* disebut juga *lateness* positif

7. *Earliness* (E_j)

   Saat penyelesaian terlalu awal yaitu sebelum *due date*. *Earliness* disebut juga *lateness* negatif.

8. *Completion time* (C_{ij})

yaitu waktu yang dibutuhkan untuk menyelesaikan suatu operasi dari pekerjaan j pada mesin i . Dalam waktu proses ini sudah termasuk waktu yang dibutuhkan untuk persiapan dan pengaturan (*set up*).

$C_j = F_j + r_j$

9. *Slack time* (S_j)

   Waktu sisa yang tersedia bagi suatu *job j*

10. *Due Date* (d_j)

    yaitu batas waktu penyelesaian yang ditentukan untuk *job j* .

11. *Makespan* (C_{max})

    Jangka waktu penyelesaian suatu penjadwalan yang merupakan waktu penyelesaian dari *job* yang berada pada urutan pengerjaan terakhir pada proses produksi .

$$C_{max} = \sum_j tj$$

12. *Waiting time* (W_j)

    yaitu waktu yang dilalui suatu pekerjaan sebelum mulai diproses

2.1.5 Kriteria Penjadwalan

Beberapa fungsi tujuan untuk mengukur kinerja optimisasi pada persoalan penjadwalan ditunjukan seperti dalam tabel berikut.

Tabel 2.1 Fungsi tujuan pada persoalan penjadwalan

Fungsi Tujuan	Notasi	Interpretasi
Makespan	C_{max}	Jumlah total waktu yang diperlukan untuk sepenuhnya memproses semua pekerjaan.
Mean completion time	$\bar{C}$	Rata-rata waktu yang dihabiskan oleh pekerjaan dalam jadwal.
Mean flow time	$\bar{F}$	Rata-rata waktu yang dihabiskan oleh pekerjaan sesuai jadwal dan termasuk waktu proses, waktu tunggu dan waktu transfer.
Maximum lateness	L_{max}	Selisih antara waktu penyelesaian dan tanggal jatuh tempo pekerjaan.
Maximum tardiness	T_{max}	Nilai maksimum keterlambatan pekerjaan.
Number of tardy jobs	N_T	Jumlah pekerjaan yang selesai setelah *due date*

Sumber: Alharkan, (2005)

2.2 *Jhop-Shop Scheduling Problem* (JSSP)

JSSP merupakan permasalahan pengalokasian beberapa mesin untuk menyelesaikan beberapa pekerjaan yang terdiri dari beberapa operasi yang telah diketahui durasi penyelesainnya. JSSP dapat didefinisikan sebagai berikut: Terdapat suatu himpunan mesin $M = \{M_1, M_2, \dots, M_m\}$ dan himpunan *job* $J = \{J_1, J_2, \dots, J_n\}$. Setiap *job* J harus melalui m mesin untuk menyelesaikan pekerjaannya. Setiap *job* terdiri dari satu himpunan operasi $O = \{O_1, O_2, \dots, O_m\}$, dan urutan operasi untuk mesin sudah ditentukan sebelumnya. Setiap operasi menggunakan salah satu mesin untuk menyelesaikan satu pekerjaan untuk interval waktu yang tetap. Permasalahan yang muncul adalah bagaimana menyusun penjadwalan operasi yang akan dikerjakan mesin dengan memenuhi kendala yang ada untuk menghasilkan kualitas solusi jadwal yang optimal (Gonçalves, *et al.*, 2005).

Tujuan utama JSSP biasanya digunakan untuk menemukan jadwal mesin terbaik untuk melayani semua pekerjaan guna mengoptimalkan kriteria tunggal / tujuan atau penjadwalan dengan multi tujuan. Fungsi tujuan ini juga dikenal sebagai suatu ukuran dari kinerja *job shop*. Ada beberapa kriteria untuk mengukur kinerja JSSP seperti minimasi *makespan, mean flow time, mean tardiness, earliness, maximum lateness* dan lain sebagainya. Yang paling sering dijumpai pada banyak penelitian adalah minimasi *makespan*. *Makespan* adalah maksimum total waktu penyelesaian dari operasi akhir dalam suatu jadwal yang terdiri dari $n \times m$ operasi.

Batasan (*constraints*) secara umum yang digunakan dalam JSSP adalah (Zaher , *et al.*, 2017):

1. Setiap pekerjaan harus diproses oleh masing-masing mesin dalam urutan tertentu (*precedence constraints*)
2. Setiap mesin hanya bisa memproses satu pekerjaan dalam satu waktu.
3. Setiap pekerjaan hanya bisa diproses oleh satu mesin dalam satu waktu
4. Setelah pekerjaan sudah mulai diproses maka tidak bisa diinterupsi.

Model matematis JSSP secara umum beserta rincian kendala ketersediaan mesin dan variabel disajikan sebagai berikut (Ploydanai & Mungwattana, 2010):

Misalkan $t_{i,j}$ adalah waktu awal *job* j yang dikerjakan pada mesin i, misalkan $f_{i,j}$ adalah waktu akhir *job* j yang dikerjakan pada mesin i, misalkan $P_{i,j}$ adalah waktu pemrosesan *job* j yang dikerjakan pada mesin i, dan misalkan (C_{max}) adalah m*akespan* (waktu penyelesaian *job* terakahir) yang dikerjakan pada mesin i. fungsi

tujuan dari penyelesaian *job* adalah untuk meminimasi *makespan* maka model matematis dari penjadwalan *job shop* seperti berikut ini:

$$\textbf{min } C_{max} \tag{2.1}$$

Sedemikian hingga.

$$t_{h,j} - t_{i,j} \geq P_{i,j} \tag{2.2}$$

$$C_{max} - t_{i,j} \geq P_{i,j} \tag{2.3}$$

$$t_{i,j} - t_{i,k} \geq P_{i,k} \quad \text{atau} \quad t_{i,k} - t_{i,j} \geq P_{i,j} \tag{2.4}$$

$$t_{i,j} \geq 0 \tag{2.5}$$

$$t_{i,j} \geq r_i \tag{2.6}$$

$$t_{i,j} + p_{i,j} \leq d_j \tag{2.7}$$

Persamaan (2.1) adalah formulasi dari fungsi tujuan yaitu meminimasi *makespan*. persamaan (2.2) digunakan untuk memastikan bahwa langkah berikutnya pada mesin h dari *job* j dimulai setelah waktu selesai dari langkah pada mesin i dari *job* j. Selanjutnya, persamaan (2.3) memastikan bahwa C_{max} harus lebih besar dari waktu selesai *job* terakhir. Persamaan (2.4) digunakan untuk pengurutan *job* pada mesin. Persamaan ini berarti bahwa hanya satu *job* yang bisa diproses oleh satu mesin dalam satu waktu. Dengan menggunakan persamaan (2.5) maka waktu mulai proses bersifat non-negatif. Persamaan (2.6) memastikan bahwa suatu *job* harus mulai setelah *released time*. Persamaan (2.7) adalah *constraint* untuk mengendalikan bahwa *job* harus selesai sebelum *due dates.*

JSSP dapat direpresentasikan dalam bentuk *Gantt chart*. *Gantt chart* merupakan digaram untuk visualisasi rencana rencana produksi dan aktual produksi (GanttHenry, 1916). Ada 2 jenis *Gantt chart* untuk representasi persoalan JSSP

yaitu *Machine Gantt chart* dan *Job Gantt chart*. *Machine Gantt chart* adalah penyusunan *Gantt chart* untuk representasi JSSP berdasarkan pada susunan mesin. *Job Gantt chart* adalah penyusunan *Gantt chart* untuk representasi JSSP berdasarkan pada susunan *job*.

Misalkan terdapat JSSP berukuran 3 *job* dan 3 mesin (3×3) seperti pada Tabel 2.4. Tabel 2.2 adalah data urutan mesin yang harus dilalui oleh setiap *job* dan Tabel 2.3 bersisi data waktu pemrosesan setiap *job* pada setiap mesin maka Tabel 2.4 adalah susunan jadwal yang terdiri dari 3 *job* dan 3 mesin dengan masing masing waktu prosesnya dalam setiap operasinya:

Tabel 2.2 Urutan mesin.

Job	Mesin		
J_1	M_1	M_2	M_3
J_2	M_1	M_3	M_2
J_3	M_2	M_1	M_3

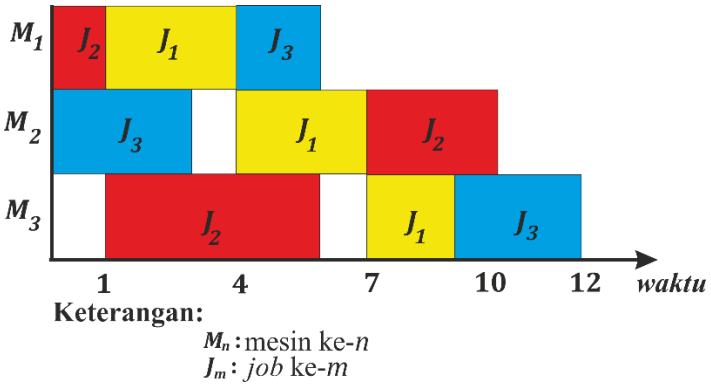
Tabel 2.3 Waktu proses.

Job	Waktu		
J_1	3	3	3
J_2	2	3	4
J_3	3	2	1

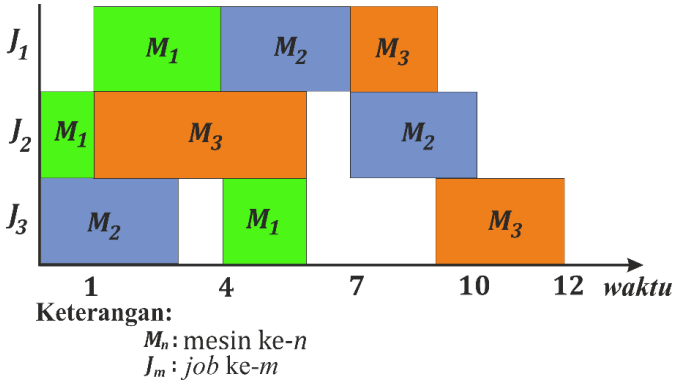
Tabel 2.4 Persoalan JSSP dengan ukuran 3×3.

Job	Operasi ke -1 Mesin (waktu proses)	Operasi ke -2 Mesin (waktu proses)	Operasi ke -3 Mesin (waktu proses)
1	1 (3)	2 (3)	3 (3)
2	1 (2)	3 (3)	2 (4)
3	2 (3)	1 (2)	3 (1)

Berdasarkan Tabel 2.4. *job* 1 akan diproses di mesin 1 selama 3 satuan waktu kemudian akan diproses di mesin 2 selama 3 satuan waktu dan terakhir akan diproses di mesin 3 selama 3 satuan waktu. *job* 2 dan *job* 3 akan diproses dimesin seperti pada *job* 1 yaitu berdasarkan urutan mesin. Representasi *Gantt chart* untuk solusi dari Tabel 2.4 dapat ditampilkan seperti berikut ini:



Gambar 2.1 *Machine Gantt chart* dari solusi JSSP ukuran 3 × 3 (modifikasi dari Gen dan Cheng, (2000)).



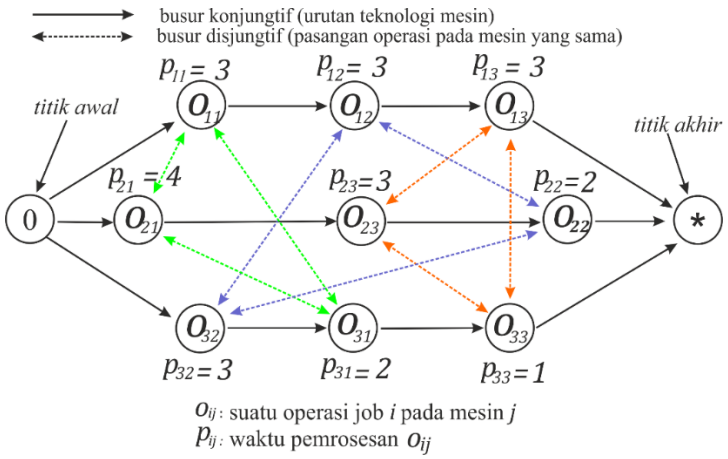
Gambar 2.2 *Job Gantt chart* dari solusi JSSP ukuran 3 × 3 (modifikasi dari Gen dan Cheng, (2000)).

.

Representasi jaringan (*network representation*) pertama kali diperkenalkan oleh Roy dan Sussman (1964). Solusi dari JSSP dapat direpresentasikan kedalam representasi jaringan yaitu berdasar pada grafik disjungtif (Balas , (1969) ; Yamada & Nakano (1997)).

Representasi solusi JSSP kedalam grafik disjungtif dapat diformulasikan dengan $V = G(C \cup D)$.

1. *V* adalah himpunan node yang merepresentasikan operasi dari masing masing *job* dengan dua node khusus yaitu node sumber (0) dan node ujung (*) yang Mewakili titik awal dan titik akhir dari jadwal.
2. C adalah himpunan busur konjungsi *(conjunctive arcs*) yang merepresentasikan urutan operasi.
3. *D* adalah himpunan busur disjungtif (*disjunctive arcs*) yang merepresentasikan pasangan operasi yang harus dilakukan pada mesin yang sama. Waktu proses untuk setiap operasi adalah bobot nilai yang melekat pada simpul yang sesuai. Representasi grafik disjuntif untuk permasalahan JSSP pada Tabel 2.4 ditunjukan pada Gambar 2.3.



Gambar 2.3 Graff disjungtif untuk JSSP berukuran 3 × 3 (Yamada & Nakano, 1997).

Berdasarkan pada Graff disjungtif pada Gambar 2.3. *job* 1 akan diproses di mesin 1 selama 3 satuan waktu, kemudian akan diproses di mesin 2 selama 3 satuan waktu dan selanjutnya diproses di mesin 3 selama 3 satuan waktu. *Job* 2 akan diproses di mesin 1 selama 4 satuan waktu kemudian akan diproses di mesin 3 selama 3 satuan waktu dan selanjutnya akan diproses di mesin 2 selama 2 satuan waktu. *Job* 3 akan diproses di mesin 2 selama 3 satuan waktu, kemudian akan diproses di mesin 1

selama 2 satuan waktu dan terakhir akan diproses di mesin 3 selama 1 satuan waktu (Yamada & Nakano, 1997).

Persoalan JSSP dengan n *job* dan m mesin maka jumlah susunan jadwal yang mungkin adalah sebesar $(n!)^m$, misalkan terdapat persoalan JSSP dengan n= 3 dan m=3 maka jumlah total urutan atau solusi yang mungkin adalah $(3!)^3 = 216$. Secara teoritis, dimungkinkan untuk menemukan susunan urutan yang optimum, namun hal ini memerlukan banyak waktu komputasi (Uzorh & Innocent, 2014).

2.3 *Benchmark Test Problem*

Untuk menemukan perbandingan yang pantas dari berbagai teknik dan algoritma, maka dibutuhkanlah pengujian pada persoalan yang sama, karenanya terbentuklah *benchmark test problem*. *benchmark test problem* memberikan standar acuan semua algoritma dapat di uji dan dibandingkan. Dengan adanya *benchmark test problem* dapat dengan mudah untuk mengukur kekuatan dan kemampuan dari berbagai algoritma. *Benchmark test problem* memiliki dimensi dan nilai tingkat kesulitan yang berbeda, hal ini memudahkan penentuan kapasitas dan keterbatasan dari metode yang diberikan untuk diuji dan juga akan memberikan informasi saran perbaikan yang dibutuhkan serta letak titik persoalan yang mesti harus diperbaiki (Jain & Meeran, 1998).

Persoalan *benchmark test problem* telah berhasil diformulasikan oleh berbagai peneliti. Beberapa *benchmark test problem* yang telah dikenal secara luas dan tersedia di OR-*library* yaitu diantaranya adalah *instance* FT06, FT10 dan FT20 diformulasikan oleh Fisher dan Thompson (1963), *instance* LA01-LA40 oleh

Lawrence (1984), *instance* ABZ5-ABZ9 oleh Adams, *et al.*, (1988), *instance* ORB01-ORB10 oleh Applegate dan Cook (1991), *instance* SWV01-SWV20 oleh Storer *et al*., (1992), *instance* YN1-YN4 oleh Yamada dan Nakano (1992) dan *instance* TA01-TA80 oleh Taillard (1993).

Benchmark test problem yang telah menerima analisis terbesar adalah yang diformulasikan oleh Fisher dan Thompson yaitu: FT06 (6×6), FT 10 (10×10), FT20 (20×5). FT06 dan FT20 telah berhasil dipecahkan secara optimal pada tahun 1975. Solusi untuk FT10 tetap sulit dipahami sampai pada tahun 1987 (Jain & Meeran, 1998).

Setiap masing masing *instance* dari *benchmark test problem* terdiri dari baris deskripsi dan baris isi yang terdiri dari nomor *job* dan nomor mesin dan setiap satu baris untuk masing masing *job* terdapat daftar nomor mesin dan waktu pemrosesan untuk setiap langkah *job*. Nomor mesin dimulai dari angka 0. Tabel 2.5 menunjukan suatu standar 6×6 *benchmark test problem* yaitu $(n = 6,\ m = 6)$ dari Fisher dan Thompson (1963) yang dikenal dengan FT06 dengan nama lain MT06.

Tabel 2.5 *Benchmark* FT06 dengan nomor mesin dimulai dari 0

Job	Operasi											
	(m,t)		*(m,t)*		*(m,t)*		*(m,t)*		*(m,t)*		*(m,t)*	
Job 1	2	1	0	3	1	6	3	7	5	3	4	6
Job 2	1	8	2	5	4	10	5	10	0	10	3	4
Job 3	2	5	3	4	5	8	0	9	1	1	4	7
Job 4	1	5	0	5	2	5	3	3	4	8	5	9
Job 5	2	9	1	3	4	5	5	4	0	3	3	1
Job 6	1	3	3	3	5	9	0	10	4	4	2	1

Pada *benchmark* di Tabel 2.5 adalah penulisan standar *benchmark test problem* JSSP pada umumnya yaitu yang diperoleh dari OR-*library*. Penomoran setiap mesin dimulai dari angka 0, namun untuk memudahkan pembacaanya beberapa

tabel *benchmark* ditulis dengan nomor mesin ditulis dengan dimulai dengan angka 1, oleh karena itu tabel *benchmark test problem* diatas ekuivalen dengan tabel *benchmark test problem* pada Tabel 2.6.

Tabel 2.6 *Benchmark test problem* FT06 dengan nomor mesin dimulai dari 1.

Job	Operasi											
	(m,t)		*(m,t)*		*(m,t)*		*(m,t)*		*(m,t)*		*(m,t)*	
Job 1	3	1	1	3	2	6	4	7	6	3	5	6
Job 2	2	8	3	5	5	10	6	10	1	10	4	4
Job 3	3	5	4	4	6	8	1	9	2	1	5	7
Job 4	2	5	1	5	3	5	4	3	5	8	6	9
Job 5	3	9	2	3	5	5	6	4	1	3	4	1
Job 6	2	3	4	3	6	9	1	10	5	4	3	1

Penjelasan *instance benchmark* diatas adalah *job* 1 harus menuju mesin 3 selama 1 satuan waktu kemudian menuju mesin 1 selama 3 satuan waktu, selanjutnya menuju mesin 2 selama 6 satuan waktu begitu terus untuk menuju langkah berikutnya seperti langkah sebelumnya. Suatu mesin tidak bisa memproses 2 *job* yang berbeda pada satu waktu. permasalahannya adalah bagaimana menyusun susunan jadwal *job* pada mesin untuk meminimasi total waktu penyelesaian yaitu total waktu diantara memulai operasi pertama hingga waktu penyelesaian operasi yang terakhir. Banyak kriteria kualitas penjadwalan yang ada tetapi *makespan* terpendek adalah yang paling sederhana dan merupakan kriteria yang secara luas paling banyak digunakan (Fang, *et al.*, 1993).

Benchmark FT06 memiliki nilai minimum *makespan* yang yang telah diketahui adalah 55 satuan waktu. Berikut disajikan nilai optimum *makespan* yang telah diketahui dari berbagai dimensi persoalan *benchmark*, yang kemudian disingkat dengan akronim BKS *(Best Know Solustion*) :

Tabel 2.7 Nilai BKS *(Best Know Solustion*) pada *benchmark test problem* JSSP

No	*Instance**	Nama Alternatif*	Dimensi $(n \times m)$*	*Best Know Solution* (BKS)**
1	FT06	MT06	6×6	55
2	FT10	MT10	10×10	930
3	FT20	MT10	20×5	1165
4	LA01	F1	10×5	666
5	LA02	F2	10×5	655
6	LA03	F3	10×5	597
7	LA04	F4	10×5	590
8	LA05	F5	10×5	593
9	LA06	G1	15×5	926
10	LA07	G2	15×5	890
11	LA08	G3	15×5	863
9	LA06	G1	15×5	926
10	LA07	G2	15×5	890
11	LA08	G3	15×5	863
12	LA09	G4	15×5	951
13	LA10	G5	15×5	958
14	LA11	H1	20×5	1222
15	LA12	H2	20×5	1039
16	LA13	H3	20×5	1150
17	LA14	H4	20×5	1292
18	LA15	H5	20×5	1207
19	LA16	A1	10×10	945
20	LA17	A2	10×10	784
21	LA18	A3	10×10	848
22	LA19	A4	10×10	842
23	LA20	A5	10×10	902
24	LA21	B1	15×10	1046
25	LA22	B2	15×10	927
26	LA23	B3	15×10	1032
23	LA20	A5	10×10	902
27	LA24	B4	15×10	935
28	LA25	B5	15×10	977
29	LA26	C1	20×10	1218

Sumber: * Beasley (1990) dan ** Gonçalves *et al* (2005)

Tabel 2.7 (Lanjutan)

No	*Instance**	Nama Alternatif*	Dimensi $(n \times m)$*	*Best Know Solution* (BKS)**
30	LA27	C2	20×10	1235
31	LA28	C3	20×10	1216
32	LA29	C4	20×10	1157
33	LA30	C5	20×10	1355
34	LA31	D1	30×10	1784
35	LA32	D2	30×10	1850
36	LA33	D3	30×10	1719
37	LA34	D4	30×10	1721
38	LA35	D5	30×10	1888
39	LA36	I1	15×15	1268
40	LA37	I2	15×15	1397
41	LA38	I3	15×15	1196
42	LA39	I4	15×15	1233
43	LA40	I5	15×15	1222

Sumber: * Beasley (1990) dan ** Gonçalves *et al* (2005)

Dua persoalan *benchmark* yang terkenal dengan ukuran 10×10 dan 20×5 yaitu FT10 (MT10) dan FT20 (MT20) umumnya digunakan sebagai *test bed* untuk mengukur keefektifan metode tertentu. MT10 biasa disebut sebagai '*notorious problem*" karena tidak bisa terpecahkan selama hingga 20 tahun, namun sekarang tidak lagi menjadi tantangan komputasi (Yamada & Nakano, 1997).

2.4 Pengertian Optimasi

Proses optimasi adalah proses mendapatkan "nilai terbaik (*best*)", jika memungkinkan untuk mengukur dan mengubah apa yang 'baik' atau 'buruk'. Dalam praktiknya, seseorang menginginkan nilai 'paling banyak' atau 'maksimal' (misalnya., Gaji) atau menginginkan nilai 'paling sedikit' atau 'minimal' (misalnya., Biaya). Oleh karena itu, kata 'optimal' diambil berarti untuk merujuk pada suatu nilai 'maksimal' atau 'minimal' bergantung pada situasinya. Kata

‘Optimal’ adalah istilah teknis yang menyiratkan pengukuran kuantitatif dan merupakan kata yang lebih tepat daripada kata ‘terbaik (*best*)’ yang lebih sesuai untuk pemakaian sehari-hari. Demikian juga, kata ‘*optimize*’, yang berarti untuk mencapai optimal, adalah kata yang lebih tepat daripada ‘meperbaiki (*improve*)’ (Antoniou & Lu, 2007). Prinsip dasar optimisasi adalah alokasi sumber daya langka yang efisien. Optimisasi dapat diterapkan pada disiplin ilmiah atau teknik apapun. Tujuan optimisasi adalah menemukan suatu algoritma yang mampu memecahkan kelas masalah tertentu. Tidak ada metode spesifik, yang mampu memecahkan semua masalah optimasi (Sivanandam & Deepa, 2008).

Gen dan Cheng (2000) mengelompokan persoalan optimisasi menjadi empat kelompok sebagai berikut:

1. Optimisasi tanpa pembatas (*unconstraint optimization*)

Optimisasi tanpa pembatas atau dikenal dengan istilah *unconstraint optimization* biasanya berkaitan dengan maksimasi atau minimisasi dari suatu fungsi dengan beberapa variabel. Setiap variabel tanpa adanya pembatas dan mempunyai nilai real.

2. Optimisasi dengan pembatas (*constraint optimization*)

Optimisasi dengan pembatas atau *constraint optimization* sering dikenal dengan istilah *nonlinear programming*. Persoalan ini berkaitan dengan optimisasi suatu fungsi tujuan yang memiliki beberapa fungsi pembatas. Fungsi pembatas ini dapat berupa suatu persamaan ataupun pertidaksamaan. Persoalan ini banyak kita jumpai dalam berbagai aplikasi teknik, operational riset, matematika, ekonomi dan lain sebagainya.

3. Optimisasi kombinatorik (*combinatorial optimization*)

Persoalan optimisasi yang mempunyai ciri bahwa terdapat solusi yang layak dalam jumlah yang terhingga, contoh Persoalan optimisasi kombinatorik adalah persoalan JSSP

4. Optimisasi dengan beberapa fungsi tujuan (*multiobjective optimization*)

Persoalan yang memiliki beberapa fungsi tujuan. Pada persoalan ini tidak selamanya terdapat solusi yang optimal untuk keseluruhan fungsi tujuan yang ada. Suatu solusi mungkin saja optimal untuk fungsi tujuan tertentu, namun sangat buruk untuk fungsi tujuan yang lain. Solusi yang demikian dikenal dengan istilah *non-dominated solution* atau *Pareto solution*.

2.5 Algoritma Genetika (*Genetic Algorithm*)

Genetic Algoritm (GA) adalah teknik optimasi pencarian acak yang menirukan proses seleksi alam (Holland, 1975). Teori evolusi pertama kalinya di perkenalkan oleh Charles Darwin melalui bukunya *"On the origin of species"* pada tahun 1859. Teori evolusi darwin mejelaskan bahwa organisme biologi akan terus ber-evolusi berdasarkan pada prinsip seleksi alam yaitu bahwa individu yang akan bertahan hidup adalah individu yang paling bisa beradaptasi terhadap lingkungannya (Darwin, 1859). Teori evolusi Darwin kemudian menginspirasi ilmuwan lain untuk mengadaptasi proses evolusi alam ke dalam berbagai persoalan, salah satunya terkait persoalan optimasi. Berdasarkam adaptasi prinsip teori evolusi alam maka

dikembangkan berbagai metode ilmiah dalam berbagai penyelasaian persoalan komputasi yaitu diantaranya *Genetic Algorithm* (GA).

GA pertama kali dikenalkan oleh Holland melalui bukunya "*Adaptation in Natural and Artificial Systems*" di tahun 1975. Holland memaparkan bagaimana mengaplikasikan prinsip prinsip evolusi alam pada persoalan optimasi dan selanjutnya membangun teori GA (Holland, 1975). GA terus dikembangkan lebih lanjut untuk lebih banyak diimplementasikan ke berbagai bidang persoalan. GA merupakan metode adaptif, yang dapat digunakan untuk penyelesaian perosalan pencarian dan optimasi (Beasley, *et al*., 1993). Terminologi yang digunakan pada proses genetika dan evolusi alam kemudian dianalogikan dalam GA. Tabel 2.8 berikut ini menjelaskan analogi terminologi dalam sistem alamiah dan terminologi dalam GA.

Tabel 2. 8 Perbandingan terminologi sistem alamiah dan terminologi GA.

Sistem Alamiah	GA
Kromosom	*String*
Gen	Fitur, Karakter, atau Detektor
Allel	Nilai fitur
Locus	Posisi *string*
Genotip	Struktur
Fenotip	kumpulan parameter, Solusi alternatif, Struktur yang di-*decode*
Epistasis	Non linieritas

Sumber: Goldberg (1989).

Saat ini GA dikenal sebagai metode yang sangat baik untuk menyelesaikan berbagai persoalan komputasi yang rumit diantaranya adalah persoalan optimasi. Meskipun GA tidak dapat menjamin bahwa solusi yang didapat akan selalu solusi global optimum. Setelah melalui proses evolusi pada beberapa generasi, GA pada umumnya akan mampu memberikan sulusi yang baik. Dalam pemanfaatan GA

untuk penyelesaian persoalan optimisasi, ada paling sedikit 3 keuntungan sebagai berikut (Syarif, 2014):

1. Pemanfaatan GA tidak memerlukan pengetahuan tentang matematika yang rumit. GA mampu menangani persoalan yang memiliki apapun fungsi tujuan dan persoalan-persoalan dengan pembatas (*constraint*) linear, nonlinear, diskrit, kontinu ataupun gabungan diantaranya.
2. Metode ini biasanya melakukan pencarian lokal terhadap solusi optimal yang terdekat. Karenanya seringkali solusi yang diperoleh tidak dapat dijamin merupakan solusi global optimal. Dengan metode ini, suatu solusi baru dapat dijamin merupakan solusi global optimal apabila persoalan yang dihadapi memiliki *convex* properti dimana setiap lokal optimal solusi dari persoalan akan juga merupakan solusi global optimal.
3. GA memberikan fleksibilitas kepada kita untuk dikombinasikan dengan metode lain (*hybrid*).

Dalam perkembangannya, GA telah diuji mampu memberikan solusi optimal dari persoalan optimisasi baik yang terdiri dari satu variabel atau multi variabel. Diantaranya GA telah diaplikasikan pada persoalan *scheduling, vehicle routing, group technology, facility layout, location allocation* dan lain sebagainya

2.5.1 Struktur GA

Secara umum algoritma genetika memiliki lima komponen dasar yaitu (Michalewicz, 1996):

1. Representasi genetik dari solusi-solusi masalah.
2. Cara membangkitkan populasi awal dari solusi-solusi.

3. Evaluasi nilai *fitness* berdasarkan kemungkinan solusi .
4. Operator-operator genetik yang merubah komposisi genetic dari *offspring* selama reproduksi.
5. Nilai-nilai untuk parameter algoritma genetika.

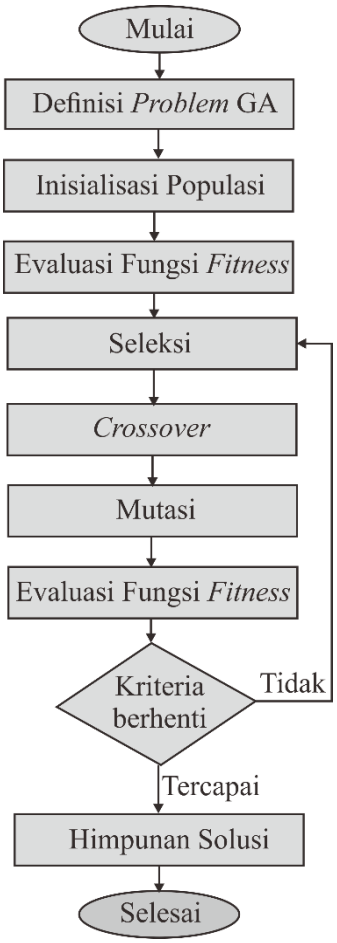
GA berbeda dengan metode optimasi konvensional dalam pencarian solusi optimal. Berikut kinerja GA dalam pencarian solusi optimal (Goldberg, 1989):

1. GA bekerja dengan proses pengkodean parameter dan GA bekerja dengan proses pengkodean solusi dari kumpulan solusi
2. GA bekerja pada sekumpulan populasi solusi bukan pada solusi tunggal
3. GA menggunakan fungsi tujuan bukan fungsi turunan.
4. GA menggunakan aturan probabilitas bukan aturan deterministik

Struktur umum untuk menjalankan GA, bisa dijelaskan dengan *psedeocode* GA Berikut ini (Gonçalves, *et al*., 2005) :

```
Genetic Algorithm {
            Bangkitkan  populasi awal P_t;
            Evaluasi  populasi awal  P_t;
            While (not Kriteria Berhenti){
                  Pilih elemen dari  P_t  salin ke P_t+1;
                  Crossover elemen dari  P_t dan  pindah ke P_t+1;
                  Mutasi elemen P_t  dan  pindah ke P_t+1;
                  Evaluasi populasi baru  P_t+1;
                  P_t = P_t+1;;
        }
}
```

psedeocode GA diatas dapat dituliskan dalam diagram alir berikut:



Gambar 2.4 Struktur GA standar

Pada GA terjadi proses iterasi sampai ditemukan nilai *fitness* terbaik pada titik kondisi pemberhentian, Proses GA dengan iterasi sampai di temukan kondisi pemberhentian dikenal dengan GA *loop* . Pada GA *loop* setiap iterasi akan terjadi proses berikut (Sivanandam & Deepa, 2008):

1. Seleksi (*selection*)

   memilih individu untuk disertakan pada proses reproduksi. Pemilihan ini dilakukan secara acak

2. Reproduksi (*reproduction*)

   Pada tahap ini dilakukan proses pembentukan kromosom baru dengan menggunskan proses rekombinasi dan mutasi

3. Evaluasi (*evaluation*)

   Dilakukannya evaluasi Nilai fitness pada masing masing individu

4. Pergantian (*replacement*)

   proses terakhir adalah dilakukannya pergantian populasi generasi tua dengan populasi generasi baru.

2.5.2 Terminologi dalam GA

Dalam GA terdapat banyak terminologi, beberapa terminologi yang digunakan diserap dari istilah bidang ilmu biologi. Berikut beberapa terminologi dalam GA (Syarif, 2014).

Kromosom	:	Kandidat solusi dari persoalan yang akan diselesaikan
Populasi	:	Sekelompok kandidat solusi yang digunakan pada algoritma genetika
Gen	:	Komponen dari kromosom (kromosom biasanya terdiri dari sekelompok gen)
Parent	:	Kromosom yang dipilih untuk proses reproduksi
Offspring	:	Kromosom baru yang diperoleh setelah proses reproduksi
Fitness	:	Suatu "nilai" yang menggambarkan kualitas dari suatu kromosom
Crossover	:	Proses reproduksi yang dilakukan dengan proses perkawinan silang antara dua kromosom
Mutasi	:	Proses reproduksi yang dilakukan dengan memodifikasi *gene* yang ada pada kromosom

2.5.3 Ruang Pencarian (*Search Spaces)*

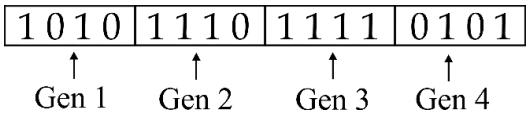
Ruang pencarian dalam optimasi mengacu pada suatu kumpulan solusi yang layak. Setiap *point* didalam ruang pencarian merepresentasikan satu kemungkinan solusi dari permasalahan yang diberikan. Dengan dikembangkannya berbagai algoritma pencarian seperti GA akan memudahkan pencarian solusi terbaik dalam ruang solusi yang luas, seperti pencarian nilai minimum atau maksimum pada persoalan optimasi (Sivanandam & Deepa, 2008)

2.5.4 Nilai *Fitness*

Nilai *fitness* individu dalam algoritma genetika adalah nilai dari fungsi tujuan untuk fenotipe nya. Nilai *fitness* digunakan sebagai acuan dalam mencapai nilai optimal dalam GA. Untuk menghitung nilai *fitness*, kromosom harus diterjemahkan (*decode)* dan fungsi tujuan harus di evaluasi. Nilai *fitness* tidak hanya mengindikasikan seberapa baik solusi yang ada tetapi juga menunjukan seberapa dekat solusi yang diberikan dengan nilai optimum (Sivanandam & Deepa, 2008). GA bertujuan untuk mencari individu dengan nilai *fitness* yang paling tinggi.

2.5.5 Gen

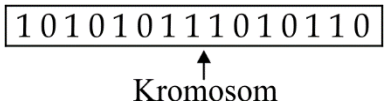
Gen adalah unit blok penyusun kromosom. Gen memiliki peran utama dalam mewarisi sifat-sifat individu. Gen merepresentasikan fitur atau karakteristik pada suatu individu. Gen adalah *string* kecil yang memiliki panjang berubah ubah. Gen dapat disusun dari kumpulan *bit, byte*, maupun *string*.



Gambar 2.5 Struktur gen (Sivanandam & Deepa, 2008)

2.5.6 Kromosom

Runtunan sejumlah gen disebut dengan kromosom (*chromosome*). Setiap kromosom dalam GA merepresentasikan kandidat solusi terhadap persoalan yang akan diselesaikan. Kromosom dapat direpresentasikan menggunakan bits, string, bilangan, Larik (*array*), *tree* ataupun objek lainnya. representasi kromosom ini sangat berpengaruh terhadap efektifitas dan efesiensi kinerja GA. Representasi kromosom yang baik harus mampu merepresentasikan semua paramater dan solusi dari persoalan. Contoh kromosom dalam konteks penjadwalan adalah urutan penyelesain job pada mesin, sedangkan dalam konteks *Travel Salesman Problem* (TSP) individu dapat menyatakan suatu kemungkinan jalur terpendek yang akan ditempuh. Gambar Berikut sebagai contoh representasi kromosom.



Gambar 2.6 Struktur kromosom (Sivanandam & Deepa, 2008).

2.5.6.1 Skema Pengkodean Kromosom

Proses merepresentasikan kromosom sebagai kandidat solusi dengan tepat, sesuai dengan persoalan yang akan diselesaikan disebut dengan proses *enconding*.Proses *enconding* mengkonversikan solusi atau kromosom kedalam bentuk kode. *Encoding* yang merupakan representasi gen suatu individu dapat diklasifikasikan berdasarkan kreteria bagaimana cara merepresentasikannya, seperti simbol yang

digunakan, struktur dari kromosom atau berdasarkan pada panjang/ukuran dari kromosom atau isi dari suatu kromosom. Ada beberapa metode *encoding* yang secara umum sering digunakan. Berikut beberapa contoh metode *encoding*:

1. Pengkodean nilai (*value encoding*)

Value encoding merupakan suatu metode representasi kromosom dengan masing masing gen langsung berisikan nilai/karakter masing-masing variabel keputusan (*decision variable*) untuk persoalan yang akan diselesaikan. Metode ini sering dipakai untuk optimasi fungsi dengan beberapa variabel, Persoalan penyesuaian kata (*word matching problem*) dan lain sebagainya.

Kromosom A	12.9755 38.7583 156.8890 45.5678 76.3888
Kromosom B	18.2735 14.5766 73.184
Kromosom C	4 8 7 4
Kromosom D	UNIVERSITAS LAMPUNG

Gambar 2.7 *Value encoding* (Syarif, 2014)

1. Pengkodean biner (*binary encoding*)

Pengkodean biner adalah suatu metode pengkodean bilangan dalam bentuk bilangan biner. setiap gen berisikan bilangan 0 atau 1. Pengkodean biner merupakan metode pengkodean yang paling umum digunakan. Metode ini sangat sederhana, mudah diciptakan dan mudah dimanipulasi. Berikut contoh kromosom yang menggunakan *binary encoding:*

Kromosom A	10001010001111101010111
Kromosom B	00000001000000011110000

Gambar 2.8 Pengkodean biner (Syarif, 2014).

2. Pengkodean permutasi (*permutation encoding*)

Pengkodean Permutasi menggunakan teknik permutasi dalam merepresentasikan kromosom. Kromosom beriisikan kumpulan nilai *integer* dengan posisi yang beurutan dan setiap nilai mewakili masing masing posisi dalam urutan nilainya. Permutation encoding bias digunakan untuk persoalan penyusunan urutan pekerjaan dan juga biasa digunakan untuk persoalan TSP (*Travelling Salesman Problem*) dan SPP (*Shortest Path Problem*).

Kromosom A	7	2	5	1	4	3	9	8	6
Kromosom B	5	2	3	1	4	9	7	6	8

Gambar 2.9 Pengkodean permutasi (Syarif, 2014).

2.5.6.2 Dekode Kromosom

Dekode kromosom adalah metode pengkodean kembali isi kromosom yang telah di *encoding*. Hasil dekode kromosom akan mewakili tiap variabel dan merepresentasikan sifat genotip dan fenotip suatu populasi.

2.5.7 Populasi (*Population*)

Populasi adalah kumpulan kromosom atau kumpulan kandidat solusi yang akan diproses dalam satu proses evolusi. Setelah mendifinisikan kromosom beserta metode representasi genetika yang digunakan, mendefinisikan parameter GA, menentukan nilai *fitnes*s dan fungsi tujuan, maka langkah selanjutnya adalah melakukan inisialisasi populasi. Populasi pada generasi awal dibangkitkan secara acak dan pada generasi selanjutnya merupakan hasil evolusi kromosom-kromosom

melalui iterasi dengan menggunakan operator *crossover* dan mutasi. Berikut contoh ilustrasi pembentukan populasi.

Populasi	Kromosom 1	11100010
	Kromosom 2	01111011
	Kromosom 3	10101010
	Kromosom 4	11001100

Gambar 2.10 Struktur populasi (Sivanandam & Deepa, 2008).

2.5.8 Seleksi

Seleksi merupakan operasi genetik pertama dalam fase reproduksi algoritma genetika. Tujuannya adalah untuk memilih Individu yang lebih baik dalam populasi yang akan menciptakan keturunan untuk generasi berikutnya (Kumar, 2012). Langkah yang harus dilakukan pada tahap ini adalah pencarian nilai *fitness* sebagai indikator kualitas individu. Pada tahap seleksi Setiap individu berkesempatan untuk dipilih. Individu yang memiliki nilai *fitness* terbaik akan memiliki peluang yang lebih besar untuk dipilih. Ada beberapa metode seleksi yang biasa digunakan yaitu diantaranya seleksi dengan metode *elitist*, seleksi berdasarkan ranking *fitness* (*rank-based fitness*), seleksi dengan roda rolet (*roulette wheel selection*), seleksi acak (*random selection*), seleksi boltzmann (*boltzmann selection*) dan seleksi dengan turnamen (*tournament selection*).

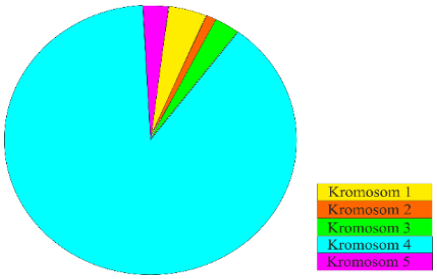
1. Seleksi roda rolet (*roulette whell selection*)

Seleksi roda rolet adalah pendekatan seleksi yang paling sederhana. Dalam metode ini semua kromosom (individu) dalam populasi ditempatkan pada roda rolet berdasarkan proporsi nilai *fitness* (Goldberg, 1989). Masing-masing individu diberi

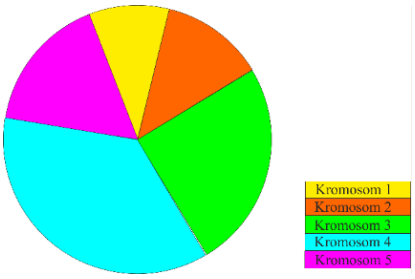
segmen roda rolet. Ukuran masing-masing segmen di roda rolet sebanding dengan nilai fitness individu. Semakin besar nilainya, semakin besar segmennya, Kemudian, roda rolet virtual diputar, Individu yang sesuai dengan segmen di mana rolet roda berhenti kemudian dipilih. Proses diulang sampai jumlah individu yang diinginkan dipilih. Individu dengan fitness lebih tinggi memiliki peluang lebih banyak untuk terpilih (Kumar, 2012).

2. *Rank selection*

Seleksi roda rolet (*roulette wheel selection*) memiliki keterbatasan berupa kendala ketika terdapat perbedaan nilai *fitness* yang sangat besar. Sebagai contoh, jika nilai *fitness* kromosom adalah 90% dari semua kromosom pada roda rolet maka kromosom lain akan memiliki sedikit peluang untuk terpilih. Dengan demikian perlu metode yang dapat mengatasi ketidak proporsional peluang masing kromosom. Metode *rank selection* mengatasi ketidak proporsionalan nilai peluang masing kromosom untuk terpilih. Metode ini menggunakan cara perangkingan nilai *fitness* pada semua kromosom. Kromosom dengan nilai *fitness* terendah diberikan nilai 1, terendah kedua diberikan nilai 2 dan *fitness* tertinggi akan diberikan nilai N (sesuai dengan jumlah kromosom pada populasi)**,** perankingan kromosom dimaksudakan agar peluang terpilih masing masing kromosom lebih proporsional. ilustrasi metode perankingan dapat dilihat seperti gambar berikut ini



Gambar 2.11 Kondisi sebelum dilakukan perangkingan.



Gambar 2.12 Kondisi setelah dilakukan perangkingan.

3. Seleksi metode *elitist*

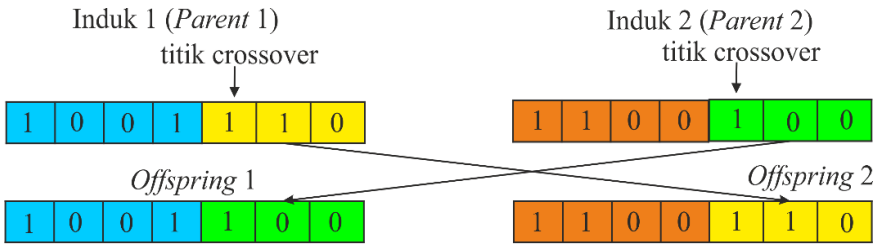
Metode *elitist* merupakan metode yang sering digunakan untuk menghindari kemungkinan kromosom dengan nilai *fitness* terbaik justru tidak terpilih ke generasi berikutnya. Metode *Elitist* dilakukan dengan pertama-tama mengambil kromosom dengan nilai *fitness* terbaik ke generasi berikutnya. Pemilihan kromosom yang lain dapat dilakukan dengan metode-metode sebelumnya. Karena metode ini dapat menghindari kemungkinan kita akan kehilangan kromosom terbaik, pada berbagai persoalan, metode ini sangatlah efektif (Syarif, 2014).

2.5.9 Crossover

Crossover adalah proses menggabungkan kromosom induk yang terpilih untuk membentuk kromosom turunan. Proses *crossover* disebut juga sebagai rekombinasi. Salah satu kelebihan GA adalah pada kemampuan pencariannya dalam keragaman solusi dengan memanfaatkan penggunaan operasi *crossover* dan mutasi. GA menerapkan mempertahankan beberapa solusi terbaik dan menghilangkan solusi yang tidak baik. Dengan operasi *crossover* diharapkan akan dihasilkan kandidat solusi baru yang lebih baik. Berikut beberapa metode *crossover* yang biasa digunakan.

1. *Crossover* satu titik (*one-point crossover*)

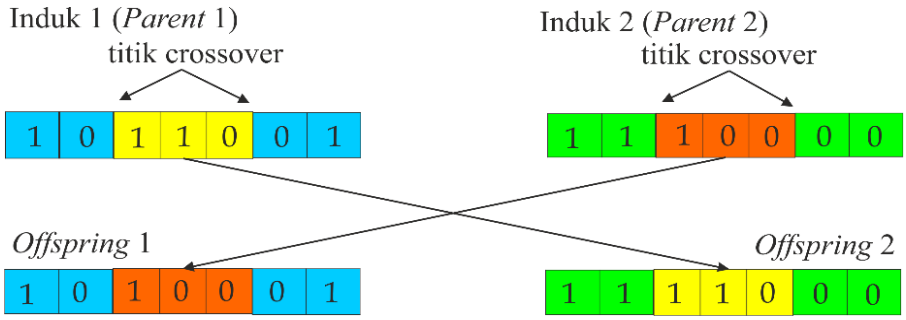
Crossover satu titik dilakukan dengan menentukan satu titik secara random pada masing masing kromosom. Proses pembentukan turunan (*offspring*) dilakukan dengan mempertukarkan elemen disebelah kanan dari titik terpilih tersebut pada masing-masing induk (*parent*). Ilustasi dari proses *crossover* satu titik dapat dilihat pada Gambar berikut.



Gambar 2.13 Contoh *crossover* satu titik (Syarif, 2014).

2. *Crossover* dua titik (*two point crossover*)

Crossover dua titik dilakukan dengan memilih dua titik secara acak pada kedua induk kromosom. Metode ini mirip dengan metode *crossover* satu titik. Proses pembentukan keturunan (*offspring*) dilakukan dengan mempertukarkan elemen diantara kedua titik pada masing masing induk (*parent*). Metode *crossover dua* titik dapat diilustrasikan seperti gambar berikut ini.



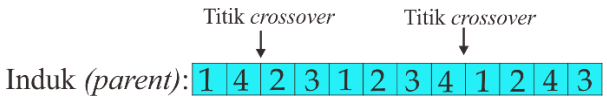
Gambar 2.14 *Crossover* dua titik (Syarif, 2014).

4. *Self crossover*

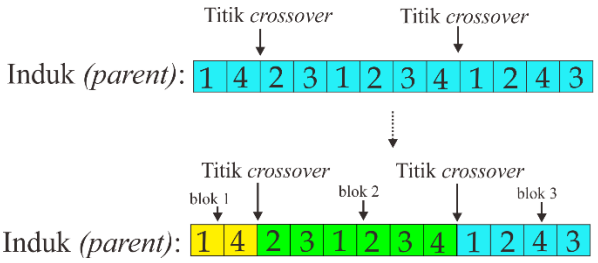
Self crossover adalah metode *crossover* yang dilakukan dengan dirinya sendiri (Hou, *et al*., 2011). Metode *crossover* ini mengadopsi cara reproduksi beberapa jenis spesies makhluk hidup yang dilakukan dengan dirinya sendiri tanpa melibatkan induk lain (*asexual reproduction*) (Sibly & Calow, 1982). Berikut prosedur metode *self crossover :*

Step 1: Pilih satu kromosom tunggal dari populasi sebagai kromosom induk (*parent*).

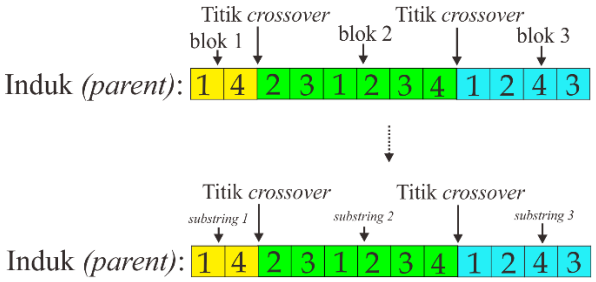
Step 2 : Tentukan 2 titik *crossover* secara acak pada kromosom induk



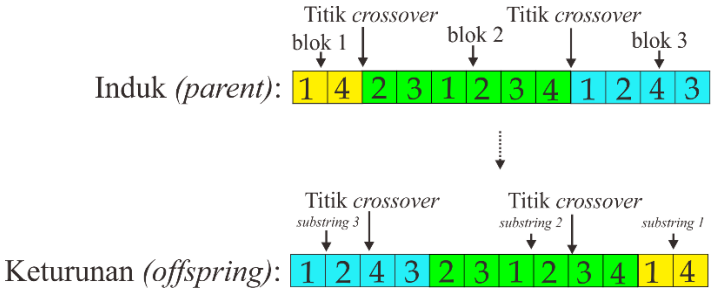
Setelah titik *crossover* di tentukan maka aka terbentuk blok diantara titik *crossover*



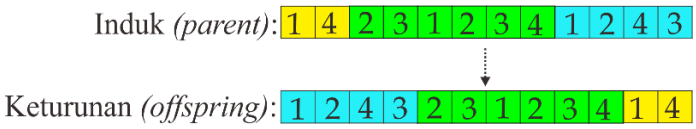
Step 3: pilih *substring* diantara 2 titik *crossover* pada kromosom induk untuk dilakukan proses *crossover.*



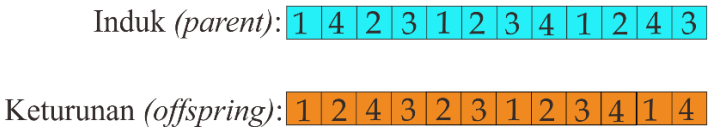
Step 4: Pindah silangkan elemen atau *substring* pada blok yang terletak diantara 2 titik *crossover* di induk kromosom ke kromosom *offspring*.



Step 5: setelah proses *crossover* selesai maka akan menghasilkan 2 individu yaitu *parent* dan *offspring*.



Step 6: Ulangi step 1 sampai step 4 untuk menghasilkan sejumlah *offspring*.

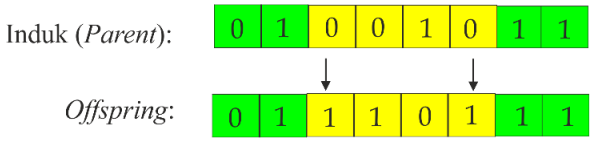


2.5.10 Mutasi

Proses mutasi dilakukan dengan melakukan perubahan terhadap gen pada suatu kromosom. Proses ini bertujuan meningkatkan keragaman kromosom yang ada pada populasi sehingga kita tidak terbawa pada optimum lokal. Proses ini dilakukan setelah dilakukannya proses *crossover*. Ada beberapa metode mutasi yang biasa digunakan antara lain:

1. Mutasi penggantian (*flip mutation*)

Flip Mutation merupakan metode yang sangat cocok untuk persoalan yang representasi kromosomnya berupa representasi biner. Metode ini dilakukan dengan cara mengubah nilai gen pada individu yang dipilih.misalkan pada kromosom induk gen bernilai 0 maka hasil mutasinya akan berubah menjadi 1, begitu pula sebaliknya.

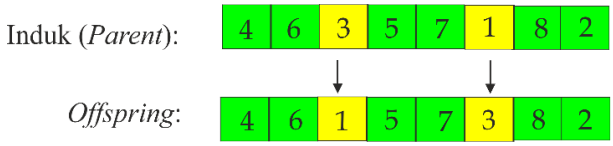


Gambar 2.15 *Flip mutation* (Syarif, 2014).

2. Mutasi pertukaran (*swap mutation*)

Swap / Exchange Mutation biasa digunakan pada kromosom yang memiliki pengkodean berdasarkan permutasi (*permutation encoding*). Pada metode *swap*

mutation, mutasi gen dilakukan dengan cara mengambil dua titik pada kromosom secara acak dan selanjutnya tukarkan masing masing gen kedua titik tersebut.



Gambar 2.16 *Swap/ Exchange mutation* (Syarif, 2014).

3. Mutasi pembalikan (*inversion mutation*)

Mutasi pembalikan (*inversion mutation*) dilakukan dengan mengambil suatu *substring* yang terletak diantara dua titik pada kromosom. Pemilihan dua titik ini dilakukan secara *random*. Selanjutnya dilakukan proses pembalikan (*invers*) gen pada *substring* tersebut:

Gambar 2.17 *Inversion mutation* (Syarif, 2014).

4. Mutasi pemindahan (*displacement mutation*)

Metode ini memilih kedua titik secara acak, ambil gen pada blok diantara dua titik tersebut, selanjutnya sisipkan kembali pada kromosom tersebut pada titik yang juga dipilih secara acak.

Gambar 2.18 *Displacement mutation* (Syarif, 2014).

5. Mutasi penyisipan (*insertion mutation*)

Metode ini hampir sama dengan mutasi pemindahan, hanya saja pada metode ini hanya dipilih satu gen untuk selanjutnya diambil dan disisipkan kembali pada kromosom, metode ini sangat mudah dan efektif seperti ilustrasi berikut.

Gambar 2.19 *Insertion mutation* (Syarif, 2014).

2.5.11 Parameter GA

Implementasi GA ke berbagai penyelesaian permasalahan pada dasarnya memiliki konsep dasar yang sama namun ada sedikit modifikasi disetiap implementasi ke permasalahan yang berbeda. Pada implementasinya GA membutuhkan parameter dasar berupa ukuran populasi (pop_size), probabilitas *crossover* (p_C), probabilitas mutasi (p_M) , maksimum generasi (max_gen). Berikut penjelasan masing masing parameter (Syarif, 2014).

1. Probabilitas persilangan (p_C)

Menunjukkan kemungkinan terjadinya *crossover* antara 2 kromosom induk. Jika tidak terjadi *crossover* atau nilai $\text{p\_C} \leq 0$ maka keturunannya akan identik dengan kromosom induk, tetapi tidak berarti generasi yang baru akan sama persis dengan generasi yang lama. Jika nilai probabilitas *crossover* atau $\text{p\_C} \geq 0$ maka beberapa kromosom akan mengalami proses *crossover* dan jika nilai $\text{p\_C} = 1$ atau 100%

maka semua kromosom akan mengalami proses *crossover*. Proses *Crossover* dilakukan dengan harapan bahwa kromosom yang baru akan lebih baik.

2. Probabilitas mutasi (p_M)

Menunjukkan seberapa sering terjadinya mutasi pada gen-gen penyusun kromosom. Jika tidak terjadi mutasi yaitu ketika nilai $p\_M \leq 0$ atau probabilitasnya 0% maka keturunan yang dihasilkan setelah *crossover* tidak berubah. Jika terjadi mutasi atau nilai $p\_M \geq 0$ maka akan terjadi perubahan bagian kromosom. Jika nilai probabilitas 100% atau nilai $p\_M = 1$ maka semua kromosom akan mengalami mutasi. Mutasi dilakukan dengan harapan bahwa kromosom yang baru akan bernilai *fitness* lebih baik dan tidak terjebak pada nilai optimum lokal.

3. Ukuran populasi (pop_size)

menunjukkan jumlah kromosom yang terdapat dalam populasi (dalam satu generasi). Jika ukuran populasi kecil maka GA akan mempunyai sedikit variasi kemungkinan untuk melakukan proses *crossover* antar kromosom induk. Sebaliknya jika terlalu banyak maka GA akan berjalan lambat.

4. Maksimum generasi (max_gen)

Menetukan jumlah populasi atau banyaknya generasi yang dihasilkan, parameter ini juga merupakan kreteria pemberhentian dari proses GA. Semakin besar nilai maksimum generasi maka akan menentukan besarnya ruang ekplorasi sehingga memungkinkan nilai yang dihasilkan bersifat *global optimum*

2.5.12 Kelebihan GA

GA sebagai alternatif penyelesaian berbagai persoalan memiliki kelebihan dan keterbatasan Beberapa hal yang termasuk kelebihan dari GA adalah sebagai berikut (Haupt & Haupt, 2004)

1. Mengoptimalkan dengan variabel kontinu atau diskrit,
2. Tidak memerlukan informasi derivatif
3. Bersamaan pencarian dari sebuah sampling yang luas pada permukaan biaya,
4. Berkaitan dengan sejumlah besar variabel,
5. Baik untuk komputer paralel.
6. Mengoptimalkan permukaan variabel dengan biaya yang sangat kompleks (GA bisa melompat dari minimum lokal).
7. Memberikan daftar variabel yang optimal, bukan hanya solusi tunggal,
8. Dapat menyandikan variabel sehingga optimasi dilakukan dengan mengkodekan variabel, dan bekerja dengan data numerik yang dihasilkan, data eksperimen, atau analitis fungsi.

2.6 Penelitian Terkait

Penelitian tentang optimisasi penyelesaian JSSP telah banyak dilakukan dan beberapa metode metode telah diaplikasikan, baik dengan pendekatan metode eksak maupun dengan pendekatan metode aproksimasi. Beberapa penelitian penelitian tentang optimisasi JSSP yang telah dilakukan diantaranya sebagai berikut:

1. *A Self-Crossover Genetic Algorithm for Job Shop Scheduling Problem* (Hou, *et al.*, 2011).

Penelitian ini menggunakan GA dengan menggunakan jenis operasi *Self-Crossover* pada tahap reproduksi GA untuk penyelesaian persoalan JSSP. Operasi *Self-Crossover* yang diusulkan pada penelitian ini diterapkan pada kromosom dengan skema penkodean permutasi dengan perulangn kromosom

2. *An Adaptive Genetic Algorithm for the Flexible Job-shop Scheduling Problem* (Pan, *et al*., 2011) .

Penelitian ini menyajikan *Adaptive Genetic Algorithm* (AGA) dan kombinasi dengan *Genetic Algorithm* (GA) untuk memecahkan persoalan *Flexible Job-shop Scheduling Problem* (FJSP).

3. *Evolutionary Algorithms for Job-Shop Scheduling* (Mesghouni & Hammadi, 2004).

Makalah ini menjelaskan bagaimana menggunakan *Evolutionary Algorithms* (EA) untuk menangani masalah *flexible Job Shop Scheduling Problem* (FJSSP) terutama untuk meminimalkan *makespan*. Tujuan dari penelitian ini adalah membangun hubungan praktis antara pengembangan di area EA dan realitas produksi JSSP.

4. *Optimization of Makespan and Mean Flow Time of Job Shop Scheduling Problem* FT06 *using* ACO (Mehmood, *et al*., 2013)**.**

Penelitian ini mengimplementasikan algoritma *Ant Colony Optimization* (ACO) untuk menyelesaikan persoalan JSSP. Hasil dari penelitian ini menunjukan bahwa penggunaaan Algoritma ACO mampu untuk menghasilkan solusi *makespan* lebih baik dan hasil komputasi kemudian dibandingkan dengan nilai *Best Know Solution*

(BKS) dari *instance benchmark* FT06. Hasil perhitungan yang diperoleh juga akan dibandingkan dengan berbagai metode pendekatan metaheuristik lain seperti *Genetic Algoritm* (GA), *Conventional Clonal Selection Algorithm* (CSA), *Positive Selection based Modified Clonal Selection Algorithm* (PSMCSA), *Particle Swarm Optimization* (PSO) dan *Tabu Search* (TS). Penelitian ini juga membahas rata rata *flow time* dan waktu komputasi dari persoalan FT06.

5. *Minimize the Makespan for Job Shop Scheduling Problem Using Artificial Immune System Approach* (Muhamad, *et al*., 2015)

Penelitian ini membahas masalah utama dalam penjadwalan *job shop* yaitu mengoptimalkan penggunaan mesin agar bisa mendapatkan waktu terpendek dalam menyelesaikan seluruh operasi. Beberapa metode telah banyak digunakan untuk menyelesaikan persoalan JSSP dan metode yang diajukan pada penelitian ini adalah dengan *Artificial Intelegent* (AI) yaitu dengan algoritma sistem kekebalan buatan (*Artificial Immune System Algorithm* / AIS). Keuntungan dari algoritma ini adalah dibuat dengan meniru sistem kekebalan alami. Hasil yang diperoleh dari penelitian dengan metode ini dibandingkan dengan hasil terbaik dari penelitian sebelumnya yaitu dari nilai *Best Know Solution* (BKS) dari *instance benchmark* JSSP.

6. *A Hybrid Bacterial Swarming Methodology for Job Shop Scheduling Environment* (Shivakumar & Amudha, 2012)**.**

Harmony Search adalah algoritma yang dirancang dengan meniru fenomena proses improvisasi musik. Dalam makalah penelitian ini, *Harmony Search* dihibridisasi dengan *Bacterial Foraging* untuk memperbaiki strategi penjadwalannya.

7. *Particle Swarm Optimization Algorithm Applied to Scheduling Problems* (Pongchairerks, 2009)

Penelitian ini menggunakan metode *Particle Swarm Optimization* (PSO) untuk penyelesaian persoalan JSSP. Perbandingan hasil uji *benchmark* dari algoritma yang diusulkan dengan algoritma yang telah ada menunjukkan bahwa algoritma yang diusulkan berkinerja lebih baik dalam beberapa kasus.

8. *A Multi-objective PSO for Job-shop Scheduling Problems* (Sha & Lin, 2009)

Sebagian besar penelitian sebelumnya mengenai masalah penjadwalan *Job-Shop* terkonsentrasi pada menemukan satu solusi optimal (misal., *Makespan*), walaupun kebutuhan aktual dari kebanyakan sistem produksi memerlukan pengoptimalan multi-tujuan. Tujuan dari penelitian ini adalah mengkonstruksi metode *Particle Swarm Optimization* (PSO) untuk persoalan penjadwalan *Job-Shop* yang rumit. Pada asalnya PSO digunakan untuk memecahkan masalah optimasi kontinu. Karena masalah optimasi penjadwalan beruang solusi diskrit, penulis memodifikasi representasi posisi partikel, pergerakan partikel, dan kecepatan partikel dalam penelitian ini. PSO yang dimodifikasi digunakan untuk memecahkan persoalan *benchmark*. Hasil pengujian menunjukkan bahwa PSO yang dimodifikasi bekerja lebih baik dalam pencarian kualitas dan efisiensi daripada metode heuristik evolusioner tradisional.

9. *A Hybrid Harmony Search Algorithm for the Job Shop Scheduling Problems* (Piroozfard, *et al.*, 2015)

Pada penelitian ini mengusulkan penggunaan *Hybrid Harmony Search Algorithm* (HHSA) yang efektif untuk menyelesaikan masalah *penjadwalan job shop* dengan

fungsi tujuan untuk meminimalkan *makespan*. Hibridasi dilakukan dengan mengkombinasikan algoritma pencarian harmoni (*Harmony Search*) dan pencarian lokal (*Local Search).* Satu set masalah *benchmark* digunakan untuk membuktikan efektivitas dan efisiensi dari algoritma yang diusulkan. Hasilnya menunjukkan bahwa algoritma HHSA yang diusulkan meningkatkan efisiensi.

10. *A Fast Taboo Search Algorithm for the Job Shop Problem* (Nowicki & Smutnicki, 1996)

Algoritma ini didasarkan pada teknik pencarian tabu (*Taboo Search*). Percobaan komputasi (sampai 2.000 operasi) menunjukkan bahwa algoritma ini tidak hanya menemukan *makespan* yang lebih pendek daripada pendekatan pendekatan terbaik lainnya tetapi juga berjalan dalam waktu yang lebih singkat. Pendektan Ini berhasil memecahkan masalah *benchmark* 10 × 10 yang merupakan persoalan terkenal dalam waktu 30 detik.

11. *A Hybrid Genetic Algorithm for Job Shop Scheduling Problem (El-Desoky , et al*., 2016)

Penelitian pada makalah ini menyajikan *Hybrid Genetic Algorithm* (HGA) untuk persoalan JSSP. Hibridasi algoritma dilakukan dengan mengkombinasikan antara *Genetic Algorithm* (GA) dan *Local Search* (LS). Pencarian lokal berdasarkan struktur lingkungan diterapkan pada hasil GA. Hasil perhitungan kemudian diuji dengan *benchmark* standar.

12. *A Hybrid Genetic Algorithm for the Job Shop Scheduling Problem* (Gonçalves, *et al.*, 2005)

Penelitian ini menyajikan *Hybrid Genetic Algorithm* (HGA) untuk persoalan JSSP. Representasi kromosom masalah didasarkan pada kunci acak (*random keys*). Jadwal dibangun menggunakan aturan prioritas di mana prioritas didefinisikan oleh *Genetic Algoritma.* Jadwal dibuat dengan menggunakan prosedur yang menghasilkan parameter jadwal aktif setelah jadwal diperoleh, kemudian pencarian lokal (*Local Search*) diterapkan untuk memperbaiki solusi. Pendekatan ini diuji pada serangkaian *benchmark* standar yang diambil dari literatur dan dibandingkan dengan pendekatan metode lain.

III. METODOLOGI PENELITIAN

3.1 Tempat dan Waktu Penelitian

Penelitian ini dilakukan di Jurusan Ilmu Komputer, Fakultas Matematika dan Ilmu Pengetahuan Alam Universitas Lampung. Waktu penelitian dilakukan selama semester ganjil tahun akademik 2017-2018.

3.2 Lingkungan Pengembangan

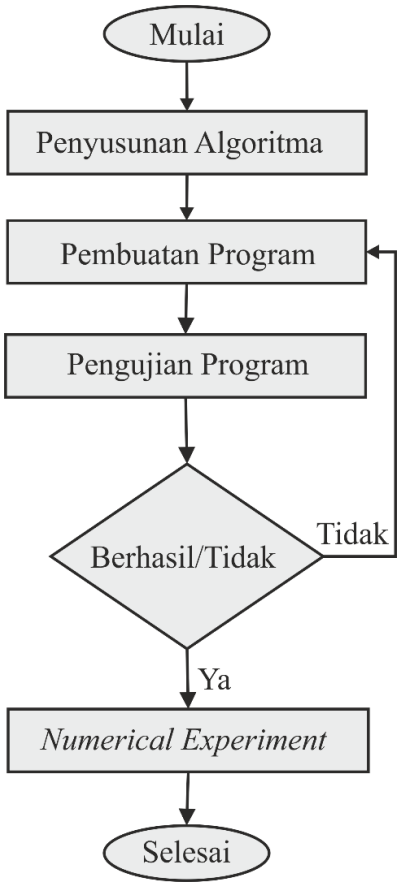
Lingkungan pengembangan yang akan digunakan pada penelitian ini adalah sebagai berikut:

1. Perangkat lunak:
    a. Sistem Operasi: Windows 8.1 Enterprise 32-bit (6.3, Build 9600)
    b. Bahasa Pemrograman: MATLAB R2015b.
2. Perangkat keras :
    a. Laptop : HP 14 Notebook PC
    b. Processor : Intel(R) Core (TM) i3-2365M CPU @1.40GHz, RAM 2 GB.

3.3 Prosedur Penelitian

Dalam merealisasikan penelitian ini maka diperlukan beberapa tahapan tahapan proses penelitian yaitu penyusunan algoritma, pembuatan program, pengujian

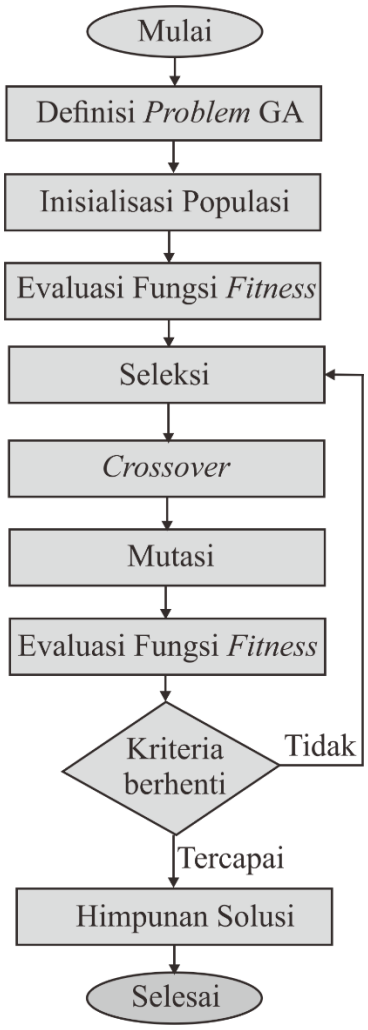
program, dan *numerical experiment*. Tahapan dalam penelitian ini secara detail dapat dijelaskan seperti pada *flowchart* prosedur penelitian berikut :



Gambar 3.1 *Flowchart* prosedur penelitian.

3.4 Algoritma dan Implementasi Program

Penyusunan model GA dan akan menjadi kerangka dasar untuk penulisan skrip program. Secara umum metode GA dapat dijelaskan dengan *flowchart* berikut ini:



Gambar 3.2 Proses kerja GA standar.

Setelah model GA disusun langkah selanjutnya adalah mengimplementasikanya kedalam program. Berikut ini prosedur pengimplementasian GA untuk penyelesaian JSSP berdasarkan pada kerangka model GA diatas.

1. Pendefinisian persoalan JSSP dalam GA

JSSP merupakan permasalahan pengalokasian beberapa mesin $M = \{M_1, M_2, \dots, M_m\}$ untuk menyelesaikan beberapa *job* $J = \{J_1, J_2, \dots, J_n\}$. yang terdiri dari beberapa operasi $O = \{O_1, O_2, \dots, O_m\}$ yang telah diketahui durasi penyelesainnya. Ada beberapa fungsi tujuan untuk penyelesaian JSSP tetapi yang

banyak dijumpai pada banyak literatur adalah minimasi *makespan*. Penilitian ini menggunakan kriteria *makespan* untuk mengukur kualitas solusi jadwal JSSP yang dihasilkan dari pencarian dengan metode GA.

Ada banyak contoh persoalan JSSP yang telah dipublikasikan oleh para peneliti dan menjadi standar persoalan JSSP didunia. Pada makalah penelitian ini kami menggunakan contoh persoalan *benchmark* JSSP dari yang di ciptakan oleh Fisher dan Thompson (1963) dan yang telah di ciptakan oleh Lawrence (Lawrence, 1984).

Berikut ini disajikan contoh persoalan JSSP dari Fisher dan Thompson (1963) yaitu *problem* FT06:

Tabel 3.1 *Instance benchmark* FT06.

Job	O_1	O_2	O_3	O_4	O_5	O_6
	m(t)	*m(t)*	*m(t)*	*m(t)*	*m(t)*	*m(t)*
Job 1	3(1)	1(3)	2(6)	4(7)	6(3)	5(6)
Job 2	2(8)	3(5)	5(10)	6(10)	1(10)	4(4)
Job 3	3(5)	4(4)	6(8)	1(9)	2(1)	5(7)
Job 4	2(5)	1(5)	3(5)	4(3)	5(8)	6(9)
Job 5	3(9)	2(3)	5(5)	6(4)	1(3)	4(1)
Job 6	2(3)	4(3)	6(9)	1(10)	5(4)	3(1)

Sumber: Fisher & Thompson (1963)

Persoalan FT06 seperti Tabel 3.1 merupakan persoalan JSSP dengan 6 *job* dan 6 mesin yang berbeda. setiap *job* diasumsikan akan diproses oleh setiap mesin tepat hanya 1 kali sehingga masing-masing *job* memiliki 6 operasi yang berbeda. Urutan operasi, penugasan mesin dan waktu pemrosesan diberikan seperti pada Tabel 3.2 berikut.

Tabel 3.2 Urutan operasi, penugasan mesin dan waktu pemrosesan pada problem FT06

Nomor *Job*	Nomor operasi dan urutan pemrosesan	Penugasan mesin	Waktu pemrosesan
J_1	O_{11}	M_3	1
J_1	O_{12}	M_1	3
J_1	O_{13}	M_2	6
J_1	O_{14}	M_4	7
J_1	O_{15}	M_6	3
J_1	O_{16}	M_5	6
J_2	O_{21}	M_2	8
J_2	O_{22}	M_3	5
J_2	O_{23}	M_5	10
J_2	O_{24}	M_6	10
J_2	O_{25}	M_1	10
J_2	O_{26}	M_4	4
J_3	O_{31}	M_3	5
J_3	O_{32}	M_4	4
J_3	O_{33}	M_6	8
J_3	O_{34}	M_1	9
J_3	O_{35}	M_2	1
J_3	O_{36}	M_5	7
J_4	O_{41}	M_2	5
J_4	O_{42}	M_1	5
J_4	O_{43}	M_3	5
J_4	O_{44}	M_4	3
J_4	O_{45}	M_5	8
J_4	O_{46}	M_6	9
J_5	O_{51}	M_3	9
J_5	O_{52}	M_2	3
J_5	O_{53}	M_5	5
J_5	O_{54}	M_6	4
J_5	O_{55}	M_1	3
J_5	O_{56}	M_4	1
J_6	O_{61}	M_2	3
J_6	O_{62}	M_4	3
J_6	O_{63}	M_6	9
J_6	O_{64}	M_1	10
J_6	O_{65}	M_5	4
J_6	O_{66}	M_3	1

Berdasarkan probem FT06 dapat dibuat matriks *M* sebagai matriks urutan mesin, dimana baris merepresentasikan urutan pemrosesan operasi dan kolom merepresentasikan urutan pemrosesan *job*. Selain itu juga dibuat matrik *T* sebagai

matrik waktu pemrosesan setiap *job*, dimana baris ke-i merepresentasikan waktu proses dari J_i untuk operasi yang berbeda.

$$M = \begin{bmatrix} M_3 & M_1 & M_2 & M_4 & M_6 & M_5 \\ M_2 & M_3 & M_5 & M_6 & M_1 & M_4 \\ M_3 & M_4 & M_6 & M_1 & M_2 & M_5 \\ M_2 & M_1 & M_3 & M_4 & M_5 & M_6 \\ M_3 & M_2 & M_5 & M_6 & M_1 & M_4 \\ M_2 & M_4 & M_6 & M_1 & M_5 & M_3 \end{bmatrix}$$

$$T = \begin{bmatrix} 1 & 3 & 6 & 7 & 3 & 6 \\ 8 & 5 & 10 & 10 & 10 & 4 \\ 5 & 4 & 8 & 9 & 1 & 7 \\ 5 & 5 & 5 & 3 & 8 & 9 \\ 9 & 3 & 5 & 4 & 3 & 1 \\ 3 & 3 & 9 & 10 & 4 & 1 \end{bmatrix}$$

Detail dari susunan *job*, urutan mesin, urutan operasi dan waktu proses seperti pada matrik mesin dan matrik waktu proses berikut ini:

	Urutan Mesin						**Waktu Proses**					
	O_1	O_2	O_3	O_4	O_5	O_6	O_1	O_2	O_3	O_4	O_5	O_6
Job 1	3	1	2	4	6	5	1	3	6	7	3	6
Job 2	2	3	5	6	1	4	8	5	10	10	10	4
Job 3	3	4	6	1	2	5	5	4	8	9	1	7
Job 4	2	1	3	4	5	6	5	5	5	3	8	9
Job 5	3	2	5	6	1	4	9	3	5	4	3	1
Job 6	2	4	6	1	5	3	3	3	9	10	4	1

Gambar 3.3 Matrik urutan mesin dan matrik waktu proses dari problem FT06.

1. Pengaturan parameter GA

Pengaturan parameter GA yaitu menentukan nilai dari parameter GA. parameter GA meliputi probabilitas persilangan (p_C), probabilitas mutasi (p_M), ukuran populasi (pop_size), dan *maximum generation* (max_gen).

2. Representasi kromosom

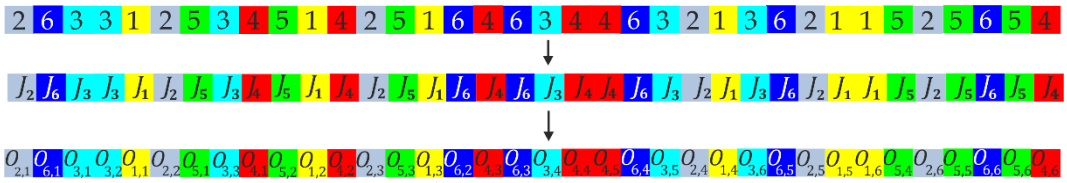
Skema pengkodean kromosom pada penelitian ini menggunakan skema pengkodean permutasi dengan perulangan (*permutation with repetition*) yang diperkenalkan oleh Bierwirth (1995). Metode representasi kromosom yang digunakan adalah metode representasi kromosom berbasis operasi (*operation-based representation*) (Zhu, *et al*., 2009). Metode representasi ini disusun berdasarkan pada pengkodean urutan operasi *job*. Setiap gen dalam kromosom mengekspresikan operasi suatu pekerjaan dan semua operasi suatu *job* ditunjukan dengan nomor urutan gen. Urutan nomor operasi pada kromosom menentukan urutan pekerjaan yang akan diproses pada mesin yang berbeda. Semua kromosom direpresentasikan kedalam bentuk array satu dimensi.

Dalam persoalan JSSP dengan sejumlah n *job* harus diproses pada sejumlah m mesin, maka jumlah gen kromosom yang terbentuk adalah jumlah semua operasi. Jika diasumsikan bahwa setiap *job* akan diproses oleh setiap mesin hanya satu kali maka panjang kromosom yaitu sejumlah $n \times m$ gen dan setiap nomor *job* akan muncul sebanyak m kali dalam 1 kromosom.

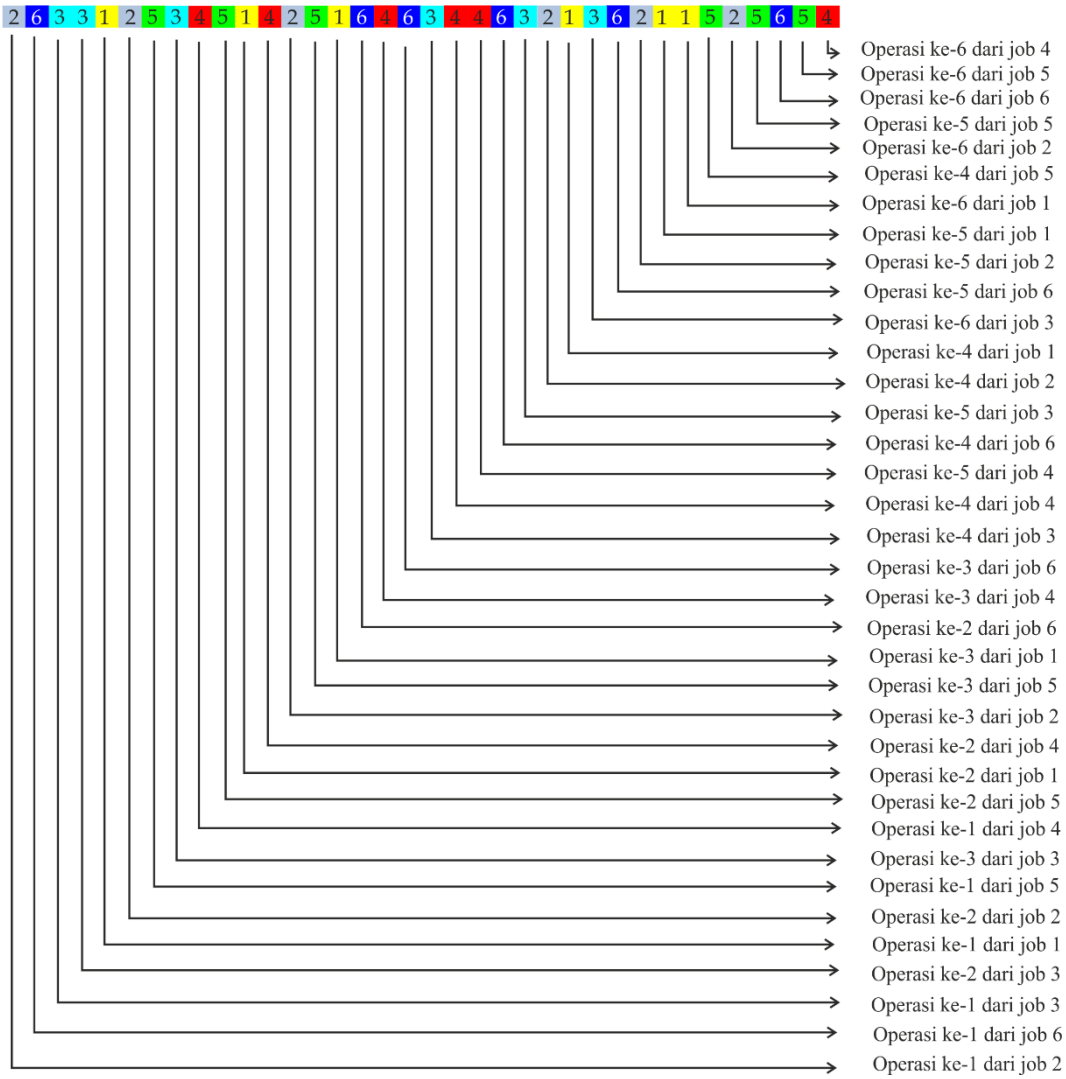
Pada metode representasi kromosom berbasis operasi, penentuan urutan operasi sesuai dengan posisi relatif gen tersebut. Urutan nomor *job* yang muncul pada waktu ke-i di dalam kromosom menunjukkan operasi ke-i di dalam kromosom tersebut. Seperti pada tabel *benchmark* FT06 yang merupakan persoalan JSSP dengan 6 *job* untuk diproses pada 6 mesin, maka contoh representasi kromosomnya adalah

Kromosom = [2 6 3 3 1 2 5 3 4 5 1 4 2 5 1 6 4 6 3 4 4 6 3 2 1 3 6 2 1 1 5 2 5 6 5 4]. Pada contoh kromosom yang diberikan tersebut, gen '2' mewakili dari *job* J_2, enam gen bernilai '2' mewakili 6 operasi dari *job* J_2, yaitu operasi ke-1, operasi ke-2, operasi ke-3, operasi ke-3, operasi ke-3, operasi ke-5 dan operasi ke-6. Gen bernilai '6' mewakili *job* J_6, enam gen bernilai '6' mewakili 6 operasi dari *job* J_6 , yaitu operasi ke-1, operasi ke-2, operasi ke-3, operasi ke-4, operasi ke-5 dan operasi ke-6. Gen bernilai 1,3,4 dan 5 memiliki prosedur teknis yang sama. Kromosom penjadwalan di atas juga dapat direpresentasikan sebagai berikut: kromosom=[$O_{2,1}$, $O_{6,1}$, $O_{3,1}$, $O_{3,2}$, $O_{1,1}$, $O_{2,2}$, $O_{5,1}$, $O_{3,3}$, $O_{4,1}$, $O_{5,2}$, $O_{1,2}$, $O_{4,2}$, $O_{2,3}$, $O_{5,3}$, $O_{1,3}$, $O_{6,2}$, $O_{4,3}$, $O_{6,3}$, $O_{3,4}$, $O_{4,4}$, $O_{4,5}$, $O_{6,4}$, $O_{3,5}$, $O_{2,4}$, $O_{1,4}$, $O_{3,6}$, $O_{6,5}$, $O_{2,5}$, $O_{1,5}$, $O_{1,6}$, $O_{5,4}$, $O_{2,6}$, $O_{5,5}$, $O_{6,6}$, $O_{5,6}$, $O_{4,6}$]. $O_{i,j}$ merupakan notasi dari operasi ke-*j* dari *job* ke-*i.* Misalnya $O_{2,1}$ adalah operasi pertama dari *job* ke-2.

Penjelasan secara rinci tentang representasi kromosom agar memudahkan memahaminya dapat dilihat seperti pada Gambar 3.4, Warna dan nomor yang sama pada gambar kromosom menunjukan nomor *job* yang sama.



Gambar 3.4 Kromosom berbasis operasi dari *problem* FT06.



Gambar 3.5 Penjelasan detail kromosom berbasis operasi dari *problem* FT06.

3. Inisialisasi populasi

Pembangkitan individu dilakukan dengan cara acak (*random*) sesuai dengan jenis representasi kromosom yang telah ditentukan. Setelah pembangkitan individu langkah yang dilakukan selanjutnya adalah menginisiasi populasi awal dengan individu yang telah terbentuk. Jumlah populasi yang terbentuk sesuai dengan parameter pop_size yang telah ditentukan sebelumnya.

4. Evaluasi Fungsi *Fitness*

Konsep nilai *fitness* adalah jika semakin besar nilai *fitness* maka akan semakin baik solusi yang didapat. Pada persoalan optimasi maksimasi semakin besar nilai optmisasi yang didapat maka nilai *fitness* semakin besar sedangkan pada persoalan optimasi minimasi adalah semakin kecil nilai optimisasi yang didapat maka nilai *fitness* akan semakin besar. Individu dengan nilai *fitness* yang lebih besar maka akan cenderung lebih berpeluang untuk maju kegenerasi berikutnya.

Salah satu nilai *fitness* yang biasa dipakai adalah dengan menghitung fungsi tujuan. Fungsi tujuan penyelesian persoalan JSSP dengan GA pada penelitian ini adalah meminimasi *makespan*. Secara matematis fungsi tujuan dapat dinyatakan seperti pada persamaan (3.1) berikut:

$$\textbf{min}\ C_{max} \tag{3.1}$$

Keterangan:

C_{max} = *Maximum completion time*

Maximum completion time adalah waktu maksimum yang dibutuhkan untuk menyelesaikan seluruh operasi pada mesin. Untuk memperoleh nilai *makespan* dapat diperoleh dengan melakukan proses *decoding* kromosom.

5. *Decoding* kromosom

Proses *decoding* kromosom merupakan proses menerjemahkan kromosom yang telah di *enconding* menjadi suatu solusi. Proses *decoding* kromosom dilakukan untuk mendapatkan nilai *makespan* dan solusi jadwal. Proses *decoding* kromosom dilakukan dengan cara mengkorespondensikan kromosom dengan matrik urutan mesin dan matrik waktu pemrosesan. Proses mengkorespondensikan kromosom

dengan matrik urutan mesin dan matrik waktu pemrosesan dilakukan dengan cara membaca nomor operasi *job* berdasarkan posisi relatif gen.

Langkah pemrosesan kromosom dilakukan berdasarkan nilai prioritas gen. Gen yang terletak diposisi sebelah kiri memiliki prioritas yang lebih besar dari yang sebelah kanan. Dengan nilai prioritas tersebut kromosom akan di proses dari sisi paling kiri. Secara umum langkah dekode kromosom untuk perhitungan *makespan* adalah seperti lberikut ini:

Langkah 1: Pilih individu atau kromosom yang akan di lakukan proses *decoding*.

Langkah 2: Baca bit kromosom berdasarkan pada prioritas yang lebih tinggi yaitu dimulai dari gen yang terletak disebelah kiri. setiap bit gen menunjukkan nomor operasi suatu *job* yang diproses pada mesin.

Langkah 3: Temukan nomor mesin di matrik urutan mesin berdasarkan informasi nomor operasi *job.*

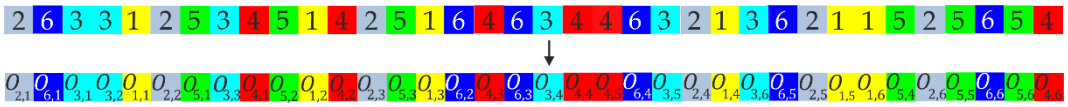
Langkah 4: Temukan nilai waktu pemrosesan di matrik waktu pemrosesan berdasarkan nomor operasi *job*.

Langkah 5: Temukan waktu maksimum dari waktu penyelesaian *job* terakhir.

Langkah 6: Perbarui waktu penyelesaian *job* saat ini dengan menambahkan waktu *job* saat ini ke hasil langkah 5;

Langkah 7: Ulangi langkah 1 ke langkah 6 sampai bit gen terakhir dari kromosom terpilih, waktu maksimum penyelesaian semua operasi *job* merupakan nilai *makespan*.

Proses *decoding* kromosom secara singkat dapat dijelaskan dengan contoh berikut: Dimisalkan terdapat kromosom dari *problem* FT06 seperti berikut.



Gambar 3.6 Kromosom dari *problem* FT06.

Proses *decoding* kromosom dimulai dari gen yang memilki prioritas paling tinggi yaitu gen yang terletak paling kiri atau dari indek kromosom pertama sampai indek kromosom terakhir.Selanjutnya korespondensikan kromosom dengan matrik urutan mesin dan matrik waktu proses berikut:

	Urutan Mesin						**Waktu Proses**					
	O_1	O_2	O_3	O_4	O_5	O_6	O_1	O_2	O_3	O_4	O_5	O_6
Job 1	3	1	2	4	6	5	1	3	6	7	3	6
Job 2	2	3	5	6	1	4	8	5	10	10	10	4
Job 3	3	4	6	1	2	5	5	4	8	9	1	7
Job 4	2	1	3	4	5	6	5	5	5	3	8	9
Job 5	3	2	5	6	1	4	9	3	5	4	3	1
Job 6	2	4	6	1	5	3	3	3	9	10	4	1

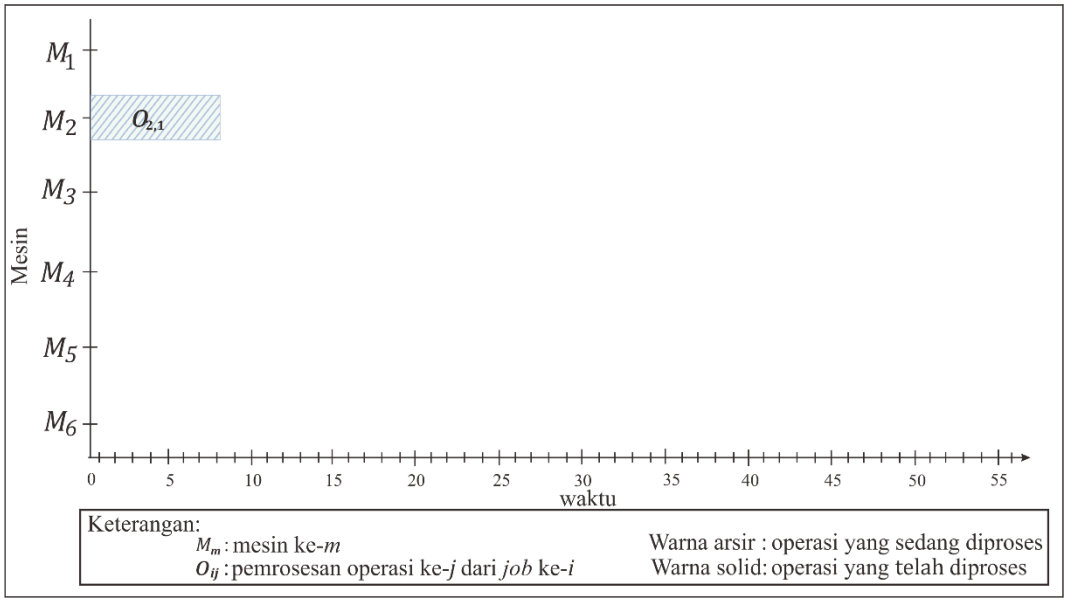
Gambar 3.7 Matrik urutan mesin dan matrik waktu proses dari *problem* FT06.

Berikut secara rinci pemrosesan kromosom dari problem FT06 yaitu pemrosesan dimulai dari gen ke-1 sampai gen ke-36 .

Gen ke-1 = O_{21}

Gen ke-1 Merupakan $O_{2,1}$ yaitu operasi ke-1 dari *job* 2. Jika dikorespondensikan berdasarkan matrik urutan mesin maka $O_{2,1} = M_2$ yang berarti operasi ke-1 dari *job* 2 akan dikerjakan di mesin 2. Jika dikorespondensikan berdasarkan matrik waktu maka $O_{2,1}$= 8 yang berarti operasi ke-1 dari *job* 2 diselesaikan selama 8 satuan waktu. Waktu awal mesin mulai bekerja menyelesaikan *job* 2 adalah di waktu 0 dan selesai di waktu 8.

Operasi ke-1 dari *job* 2 dikerjakan oleh mesin 2 selama 8 satuan waktu dapat dilihat seperti dalam *Gantt chart* berikut.

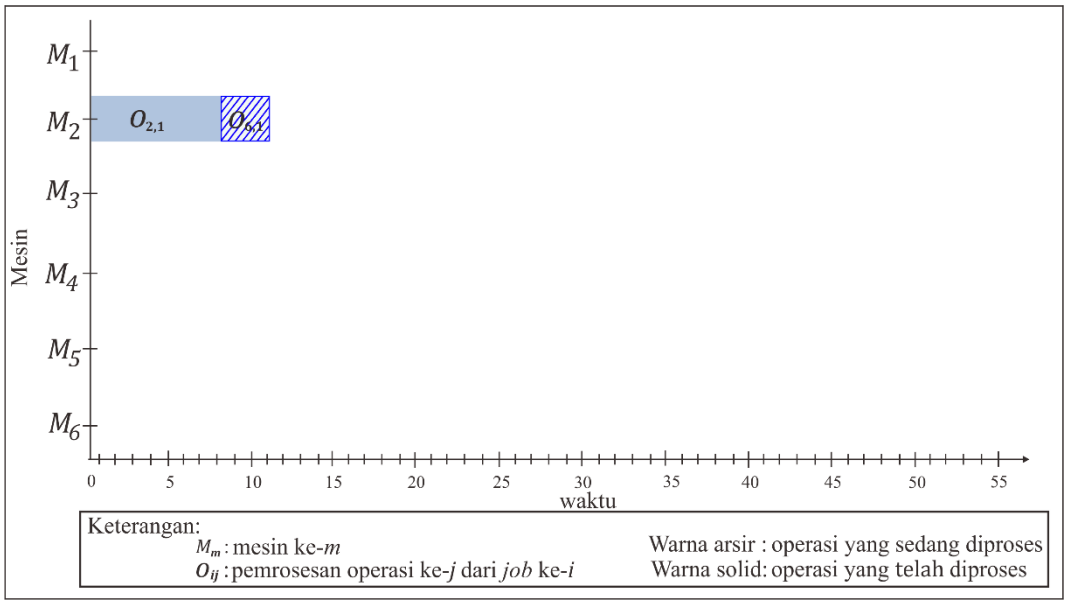


Gambar 3.8 Operasi ke-1 dari *job* 2 dikerjakan oleh mesin 2 selama 8 satuan waktu.

Gen ke-2 = O_{61}

Gen ke-2 merupakan O_{61} yaitu operasi ke-1 dari *job* 6. Jika dikorespondensikan Berdasarkan matrik mesin maka $O_{61} = M_2$, yang berarti operasi ke-1 dari *job* 6 akan dikerjakan oleh mesin 2. Jika dikorespondensikan Berdasarkan matrik waktu maka $J_{61} = 3$ yang berarti operasi ke-1 dari *job* 6 akan diselesaikan selama 3 satuan waktu. Mesin 2 tidak langsung bekerja menyelesaikan operasi ke-1 dari *job* 6 karena mesin 2 sedang mengerjakan operasi ke-1 dari *job* 2 sehingga harus menunggu sampai operasi ke-1 dari *job* 2 selesai di waktu ke 8. Waktu awal mesin 2 mulai bekerja menyelesaikan operasi ke-1 dari *job* 6 adalah di waktu ke-8 dan selesai di waktu ke- 11.

Operasi ke-1 dari *job* 6 dikerjakan oleh mesin 2 selama 3 satuan waktu dapat dilihat seperti dalam *Gantt chart* berikut.

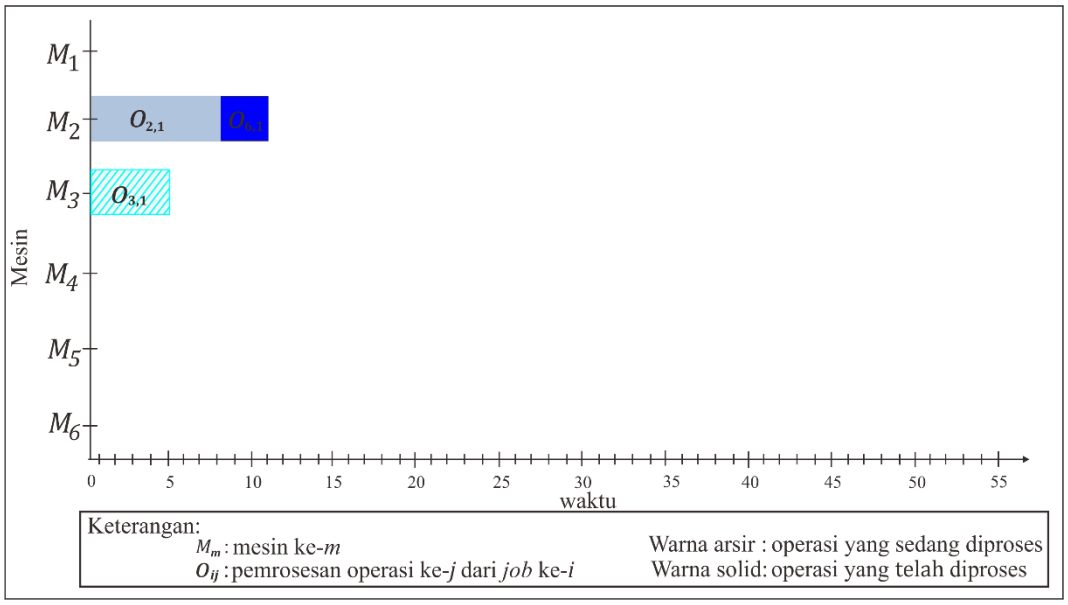


Gambar 3.9 Operasi ke-1 dari *job* 6 dikerjakan oleh mesin 2 selama 3 satuan waktu.

Gen ke-3 = O_{31}

Gen ke-3 merupakan $O_{3,1}$ yaitu *job* 3 operasi ke-1. Jika dikorespondensikan Berdasarkan matrik mesin maka $O_{3,1} = M_3$, yang berarti mesin 3 akan bekerja menyelesaikan operasi ke-1 dari *job* 3. Jika dikorespondensikan Berdasarkan matrik waktu maka J_{31}= 3, yang berarti *job* 3 operasi ke-1 akan diselesaikan selama 5 satuan waktu. Waktu awal mesin 3 mulai bekerja menyelesaikan operasi ke-1 dari *job* 3 adalah di waktu ke-0 dan selesai di waktu ke- 5.

Operasi ke-1 dari *job* 3 dikerjakan oleh mesin 3 selama 5 satuan waktu dapat dilihat seperti dalam *Gantt chart* berikut.

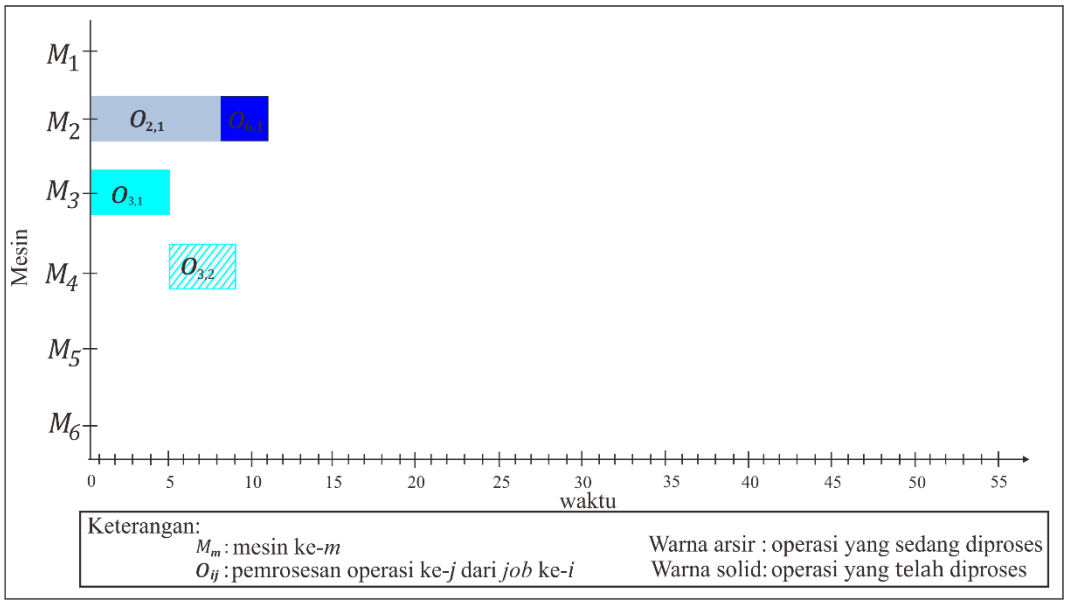


Gambar 3.10 Operasi ke-1 dari *job* 3 dikerjakan oleh mesin 3 selama 5 satuan waktu.

Gen ke-4 = O_{32}

Gen ke-4 merupakan O_{32} yaitu *job* 3 operasi ke-2. Jika dikorespondensikan Berdasarkan matrik mesin maka $O_{3,2} = M_4$, yang berarti mesin 4 akan bekerja menyelesaikan operasi ke-2 dari *job* 3. Jika dikorespondensikan Berdasarkan matrik waktu maka $O_{3,2} = 4$, yang berarti *job* 3 operasi ke-2 akan diselesaikan selama 4 satuan waktu. Dalam 1 waktu mesin tidak bisa mengerjakan *job* yang sama maka *job* 3 operasi ke-2 akan dikerjakan setelah *job* 3 operasi ke-1 yang sedang dikerjakan oleh mesin 3 terlebih dahulu selesai. Waktu awal mesin 4 mulai bekerja menyelesaikan operasi ke-2 dari *job* 3 adalah di waktu ke-5 dan selesai di waktu ke- 9.

Operasi ke-2 dari *job* 3 dikerjakan oleh mesin 4 selama 4 satuan waktu dapat dilihat seperti dalam *Gantt chart* berikut.

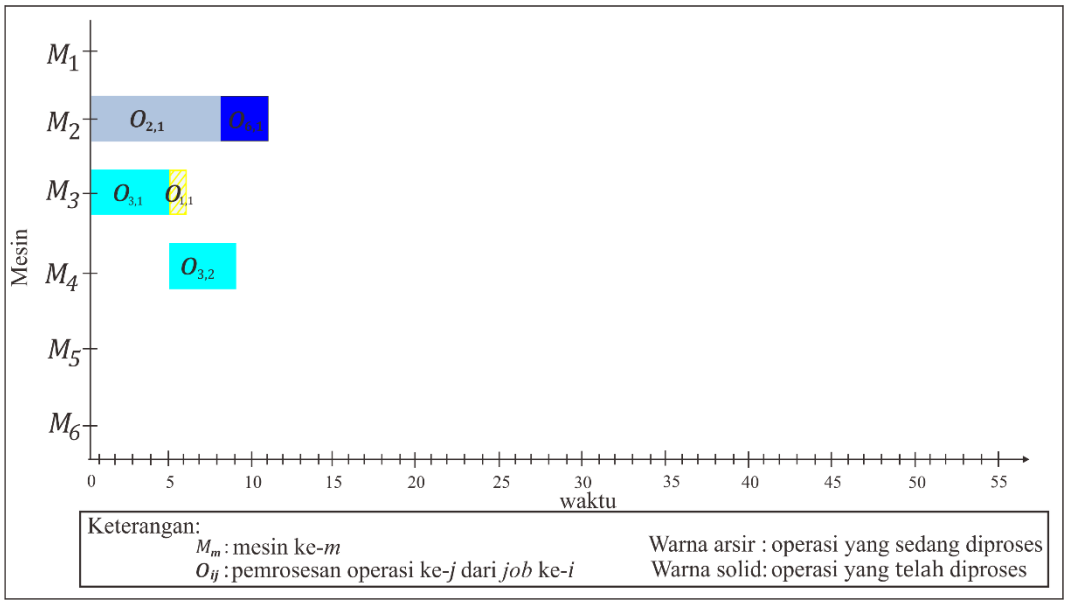


Gambar 3.11 Operasi ke-2 dari *job* 3 dikerjakan oleh mesin 4 selama 4 satuan waktu.

Gen ke-5 = $O_{1,1}$

Gen ke-5 adalah $O_{1,1}$ yaang merupakan *job* 1 operasi ke-1. Jika dikorespondensikan Berdasarkan matrik mesin maka $O_{1,1} = M_3$, yang berarti mesin 3 akan bekerja menyelesaikan operasi ke-1 dari *job* 1. Jika dikorespondensikan Berdasarkan matrik waktu maka $O_{1,1} = 1$, yang berarti *job* 1 operasi ke-1 akan diselesaikan selama 1 satuan waktu. Mesin 3 tidak langsung bekerja menyelesaikan operasi ke-1 dari *job* 1 karena mesin 3 sedang mengerjakan operasi ke-1 dari *job* nomor 3 sehingga harus menunggu sampai operasi ke-1 dari *job* 3 selesai pada waktu ke 5. Waktu awal mesin 2 mulai bekerja menyelesaikan operasi ke-1 dari *job* 1 adalah di waktu ke-5 dan selesai di waktu ke-6.

Operasi ke-1 dari *job* 1 dikerjakan oleh mesin 3 selama 1 satuan waktu dapat dilihat seperti dalam *Gantt chart* berikut.

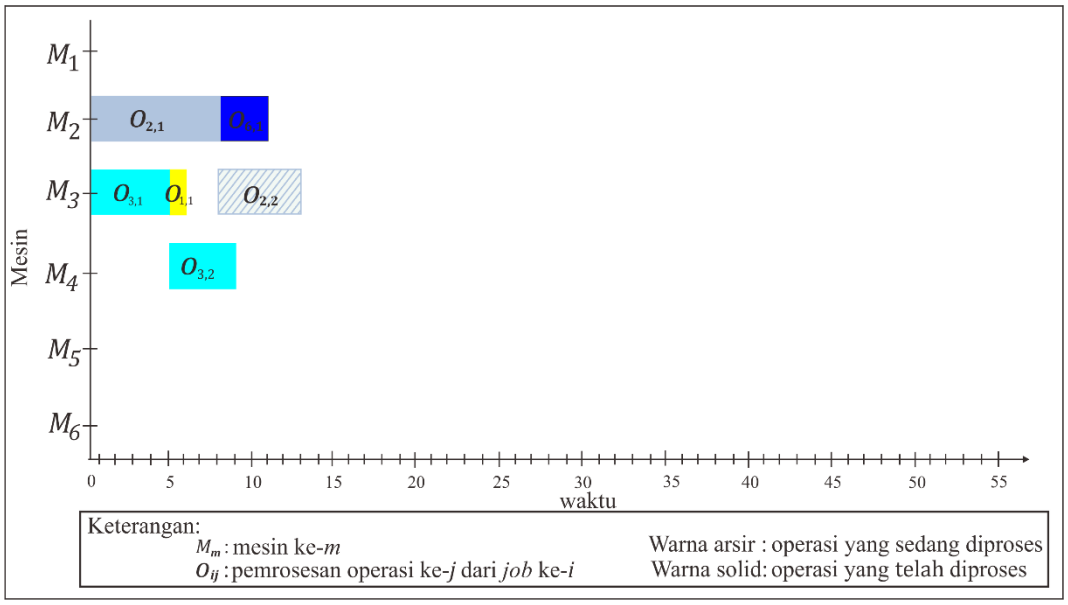


Gambar 3.12 Operasi ke-1 dari *job* 1 dikerjakan oleh mesin 3 selama 1 satuan waktu.

Gen ke-6 = $O_{2,2}$

Gen ke-6 adalah $O_{2,2}$ yang merupakan *job* 2 operasi ke-2. Jika dikorespondensikan berdasarkan matrik mesin maka $O_{2,2} = M_3$, yang berarti mesin 3 akan bekerja menyelesaikan operasi ke-2 dari *job* 2. Jika dikorespondensikan berdasarkan matrik waktu maka $O_{2,2} = 5$, yang berarti *job* 2 operasi ke-2 akan diselesaikan selama 5 satuan waktu. Dalam 1 waktu mesin tidak bisa mengerjakan *job* yang sama maka *job* 2 operasi ke-2 akan dikerjakan setelah *job* 2 operasi ke-1 yang sedang dikerjakan oleh mesin 2 terlebih dahulu selesai. Waktu awal mesin 3 mulai bekerja menyelesaikan operasi ke-2 dari *job* 2 adalah di waktu ke-8 dan selesai di waktu ke- 13.

Operasi ke-2 dari *job* 2 dikerjakan oleh mesin 3 selama 5 satuan waktu dapat dilihat seperti dalam *Gantt chart* berikut.

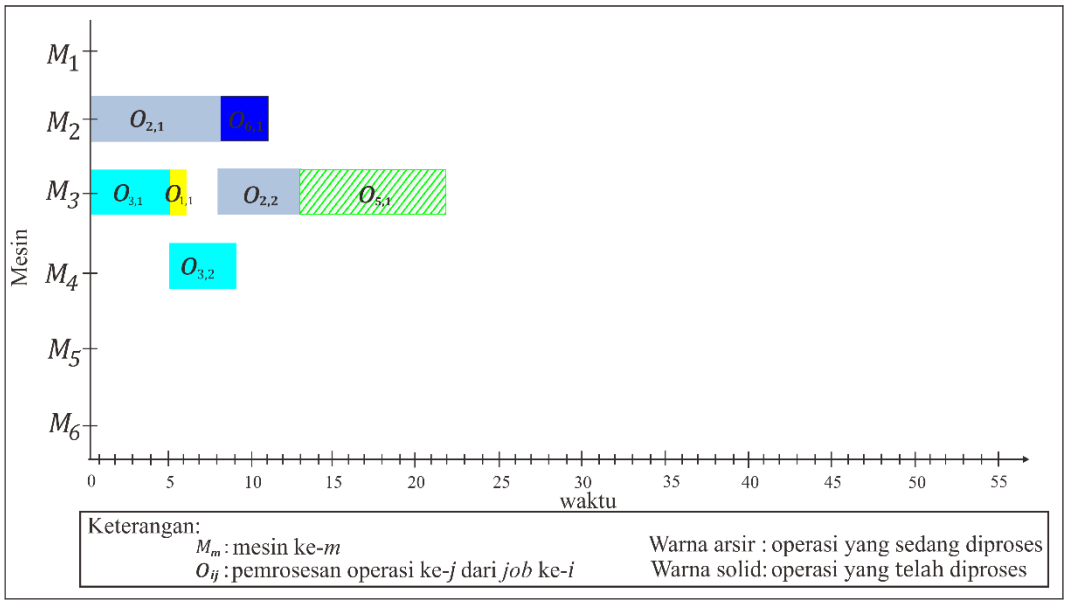


Gambar 3.13 Operasi ke-2 dari *job* 2 dikerjakan oleh mesin 4 selama 5 satuan waktu.

Gen ke-7= $O_{5,1}$

Gen ke-7 adalah $O_{5,1}$ yang merupakan *job* 5 operasi ke-1. Jika dikorespondensikan berdasarkan matrik mesin maka $O_{5,1} = M_3$, yang berarti operasi ke-1 dari *job* 5 akan diselesaikan oleh mesin 3. Jika dikorespondensikan berdasarkan matrik waktu maka $O_{5,1}$= 9 , yang berarti operasi ke-1 dari *job* 5 akan diselesaikan selama 9 satuan waktu. Mesin 3 akan bekerja menyelesaikan operasi ke-1 dari *job* 5 setelah operasi ke-2 dari *job* 2 selesai yaitu pada waktu ke-13. Waktu awal mesin 3 bekerja menyelesaikan operasi ke-1 dari *job* 5 adalah di waktu ke-13 dan selesai di waktu ke- 22.

Operasi ke-1 dari *job* 5 dikerjakan oleh mesin 3 selama 9 satuan waktu dapat dilihat seperti dalam *Gantt chart* berikut.

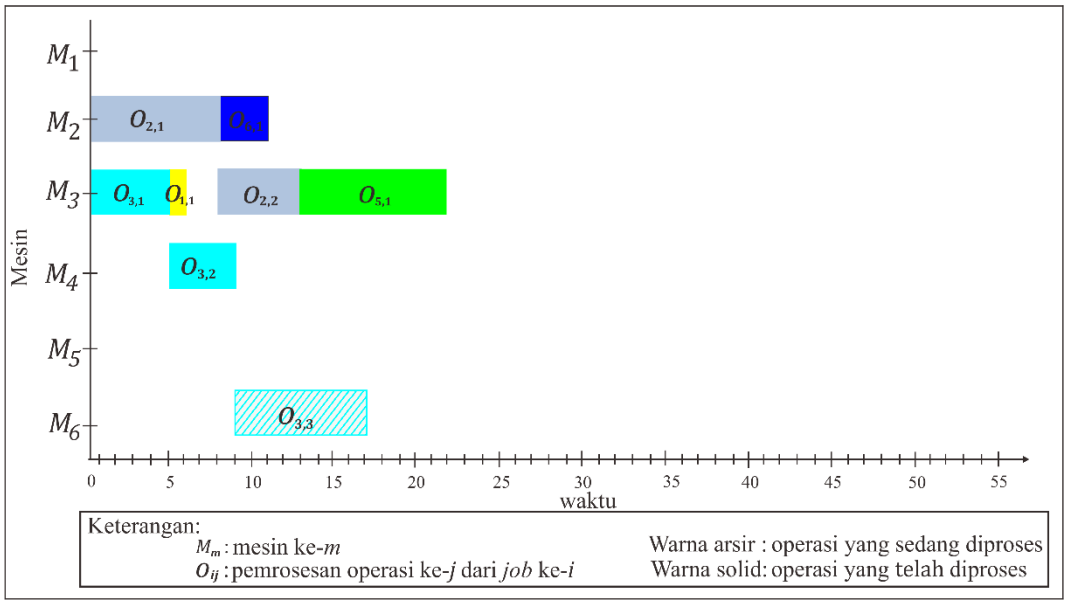


Gambar 3.14 Operasi ke-1 dari *job* 5 dikerjakan oleh mesin 3 selama 9 satuan waktu.

Gen ke-8 = $O_{3,3}$

Gen ke-8 adalah $O_{3,3}$ yang merupakan *job* 3 operasi ke-3. Jika dikorespondensikan berdasarkan matrik mesin maka $O_{3,3} = M_6$, yang berarti operasi ke-3 dari *job* 3 akan diselesaikan oleh mesin 6. Jika dikorespondensikan berdasarkan matrik waktu $O_{3,3}$= 8, yang berarti operasi ke-3 dari *job* 3 akan diselesaikan selama 8 satuan waktu. Mesin 6 akan bekerja menyelesaikan operasi ke-3 dari *job* 3 setelah selesai operasi ke-2 dari *job* 3 yang dikerjakan oleh mesin 4 yaitu pada waktu ke-9. Waktu awal mesin 6 bekerja menyelesaikan operasi ke-3 dari *job* 3 adalah di waktu ke-9 dan selesai di waktu ke- 17.

Operasi ke-3 dari *job* 3 dikerjakan oleh mesin 6 selama 8 satuan waktu dapat dilihat seperti dalam *Gantt chart* berikut.

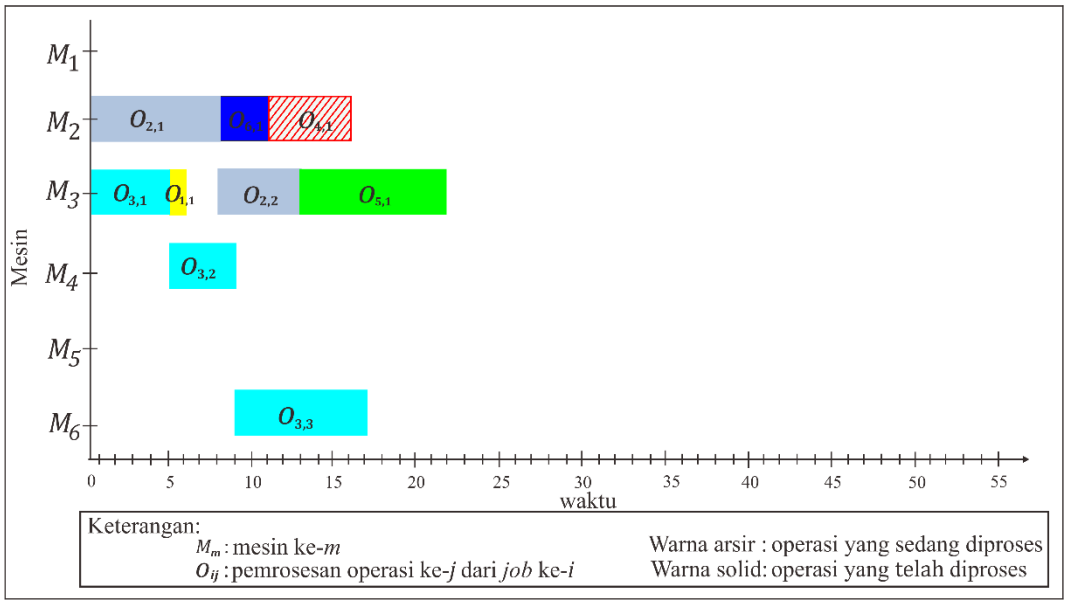


Gambar 3.15 Operasi ke-3 dari *job* 3 dikerjakan oleh mesin 6 selama 8 satuan waktu.

Gen ke-9 = $O_{4,1}$

Gen ke-9 adalah $O_{4,1}$ yang merupakan *job* 4 operasi ke-1. Jika dikorespondensikan berdasarkan matrik mesin maka $O_{4,1} = M_2$, yang berarti operasi ke-1 dari *job* 4 akan diselesaikan oleh mesin 2. Jika dikorespondensikan berdasarkan matrik waktu maka $O_{4,1} = 5$, yang berarti operasi ke-1 dari *job* 4 akan diselesaikan selama 5 satuan waktu. Mesin 2 akan bekerja menyelesaikan operasi ke-1 dari *job* 4 setelah operasi ke-1 dari *job* 6 selesai pada waktu ke-11. Waktu awal mesin 3 bekerja menyelesaikan operasi ke-1 dari *job* 4 adalah di waktu ke-11 dan selesai di waktu ke- 16.

Operasi ke-1 dari *job* 4 dikerjakan oleh mesin 2 selama 5 satuan waktu dapat dilihat seperti dalam *Gantt chart* berikut.

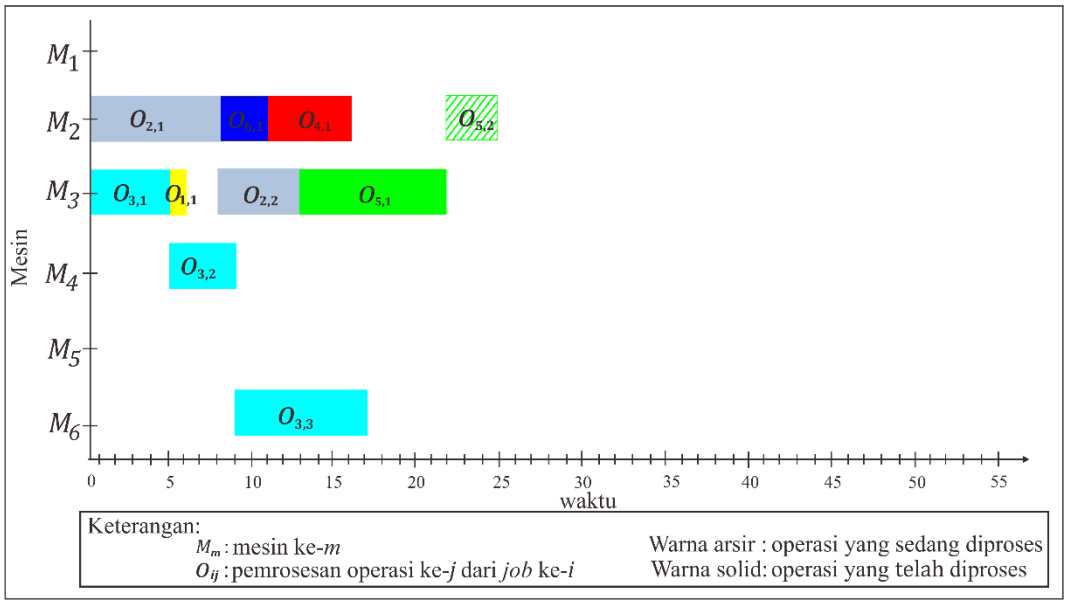


Gambar 3.16 Operasi ke-1 dari *job* 4 dikerjakan oleh mesin 2 selama 5 satuan waktu.

Gen ke-10 = $O_{5,2}$

Gen ke-10 merupakan $O_{5,2}$ yaitu operasi ke-1 dari *job* 5. Jika dikorespondensikan berdasarkan matrik mesin maka $O_{5,2} = M_2$, yang berarti operasi ke-2 dari *job* 5 akan diselesaikan oleh mesin 2. Jika dikorespondensikan berdasarkan matrik waktu maka $O_{5,2}$= 3, yang berarti operasi ke-2 dari *job* 5 akan diselesaikan selama 3 satuan waktu. Mesin 2 akan bekerja menyelesaikan operasi ke-2 dari *job* 5 setelah operasi ke-1 dari *job* 5 selesai dikerjakan oleh mesin 3 di waktu ke-22. Waktu awal mesin 2 bekerja menyelesaikan operasi ke-2 dari *job* 5 adalah di waktu ke-22 dan selesai di waktu ke- 25.

Operasi ke-2 dari *job* 5 dikerjakan oleh mesin 2 selama 3 satuan waktu dapat dilihat seperti dalam *Gantt chart* berikut.

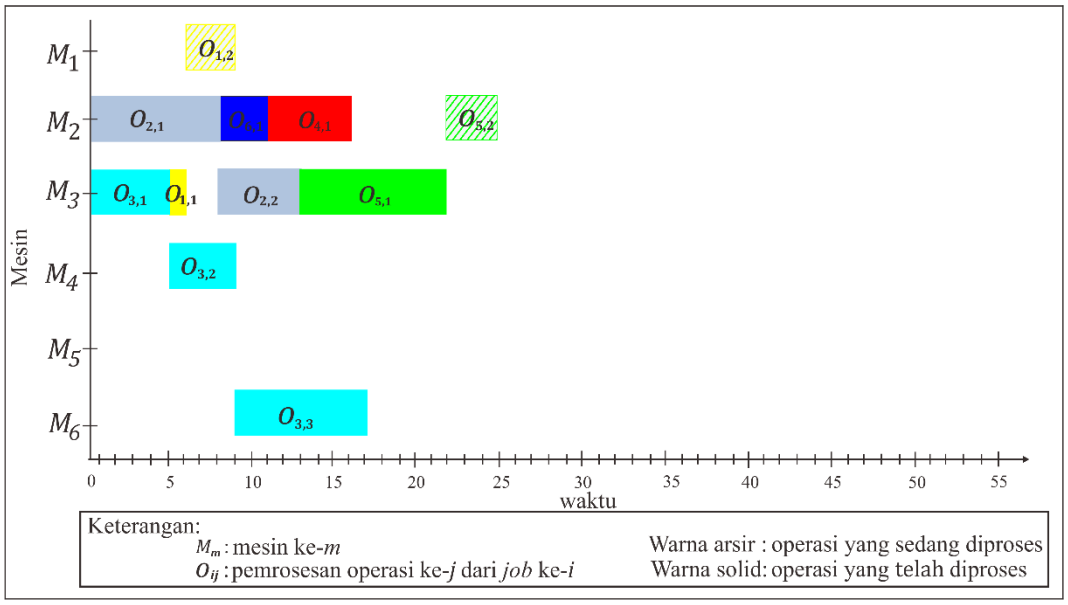


Gambar 3.17 Operasi ke-2 dari *job* 5 dikerjakan oleh mesin 2 selama 3 satuan waktu.

Gen ke-11 = $O_{1,2}$

Gen ke-11 merupakan $O_{1,2}$ yaitu operasi ke-2 dari *job* 1. Jika dikorespondensikan berdasarkan matrik mesin maka $O_{1,2}$ =M_1, yang berarti operasi ke-2 dari *job* 1 akan diselesaikan oleh mesin 1. Jika dikorespondensikan berdasarkan matrik waktu maka $O_{1,2}$ = 3 , yang berarti operasi ke-2 dari *job* 1 akan diselesaikan selama 3 satuan waktu. Mesin 1 akan bekerja menyelesaikan operasi ke-2 dari *job* 1 setelah operasi ke-1 dari *job* 1 selesai dikerjakan oleh mesin 3 di waktu ke-6. Waktu awal mesin 1 bekerja menyelesaikan operasi ke-2 dari *job* 1 adalah di waktu ke-6 dan selesai di waktu ke- 9.

Operasi ke-2 dari *job* 1 dikerjakan oleh mesin 1 selama 3 satuan waktu dapat dilihat seperti dalam *Gantt chart* berikut.

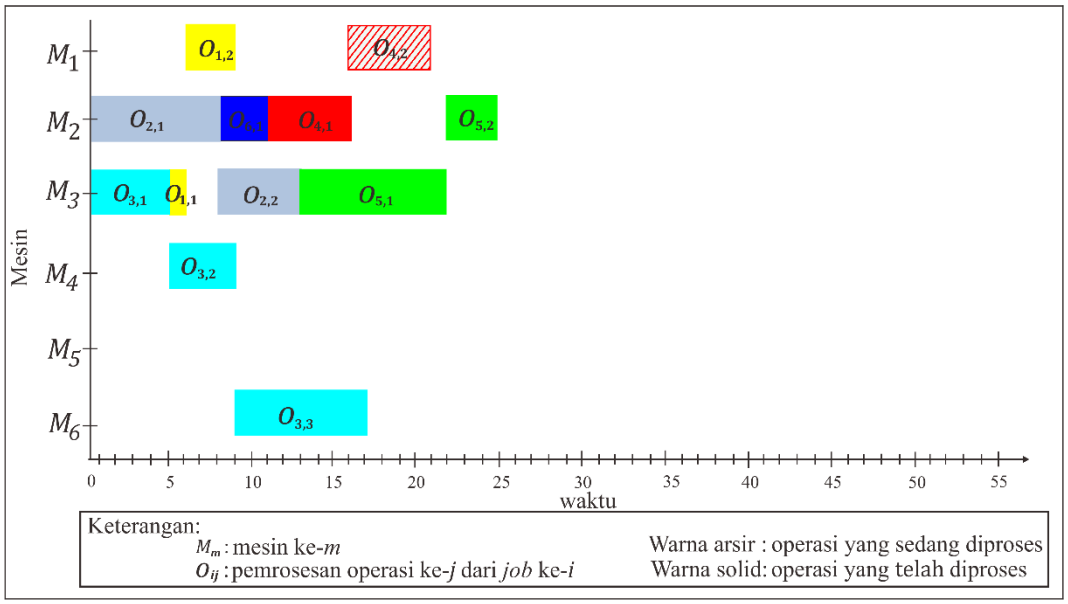


Gambar 3.18 Operasi ke-2 dari *job* 1 dikerjakan oleh mesin 1 selama 3 satuan waktu.

Gen ke-12 = $O_{4,2}$

Gen ke-12 merupakan $O_{4,2}$ yaitu operasi ke-2 dari *job* 4. Jika dikorespondensikan berdasarkan matrik mesin maka $O_{4,2} = M_1$, yang berarti operasi ke-2 dari *job* 4 akan diselesaikan oleh mesin 1. Jika dikorespondensikan berdasarkan matrik waktu maka $O_{4,2} = 5$, yang berarti operasi ke-2 dari *job* 4 akan diselesaikan selama 5 satuan waktu. Mesin 1 akan bekerja menyelesaikan operasi ke-2 dari *job* 4 setelah operasi ke-1 dari *job* 4 selesai dikerjakan oleh mesin 2 di waktu ke-13. Waktu awal mesin 1 bekerja menyelesaikan operasi ke-1 dari *job* 1 adalah di waktu ke-13 dan selesai di waktu ke- 18.

Operasi ke-2 dari *job* 4 dikerjakan oleh mesin 1 selama 5 satuan waktu dapat dilihat seperti dalam *Gantt chart* berikut.

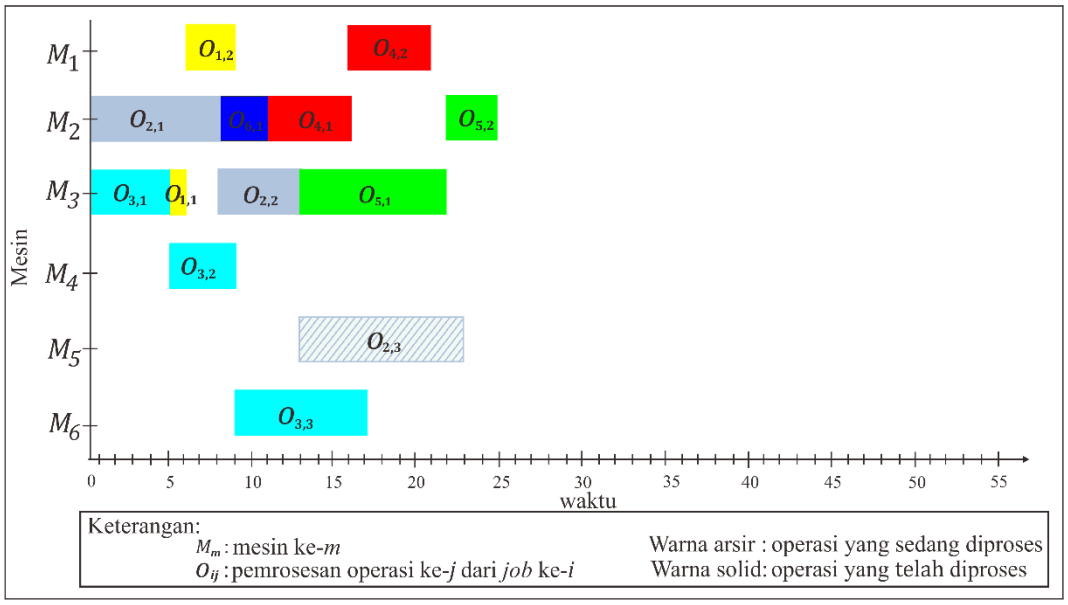


Gambar 3.19 Operasi ke-2 dari *job* 4 dikerjakan oleh mesin 1 selama 5 satuan waktu.

Gen ke-13 = $O_{2,3}$

Gen ke-13 merupakan $O_{2,3}$ yaitu operasi ke-3 dari *job* 2. Jika dikorespondensikan berdasarkan matrik mesin maka $O_{2,3} = M_5$, yang berarti operasi ke-3 dari *job* 2 akan diselesaikan oleh mesin 5. Jika dikorespondensikan berdasarkan matrik waktu maka $O_{2,3} = 10$, yang berarti operasi ke-3 dari *job* 2 akan diselesaikan selama 10 satuan waktu. Mesin 1 akan bekerja menyelesaikan operasi ke-3 dari *job* 2 setelah operasi ke-2 dari *job* 2 selesai dikerjakan oleh mesin 3 di waktu ke-13. Waktu awal mesin 5 bekerja menyelesaikan operasi ke-3 dari *job* 2 adalah di waktu ke-13 dan selesai di waktu ke- 23.

Operasi ke-3 dari *job* 2 dikerjakan oleh mesin 5 selama 10 satuan waktu dapat dilihat seperti dalam *Gantt chart* berikut.

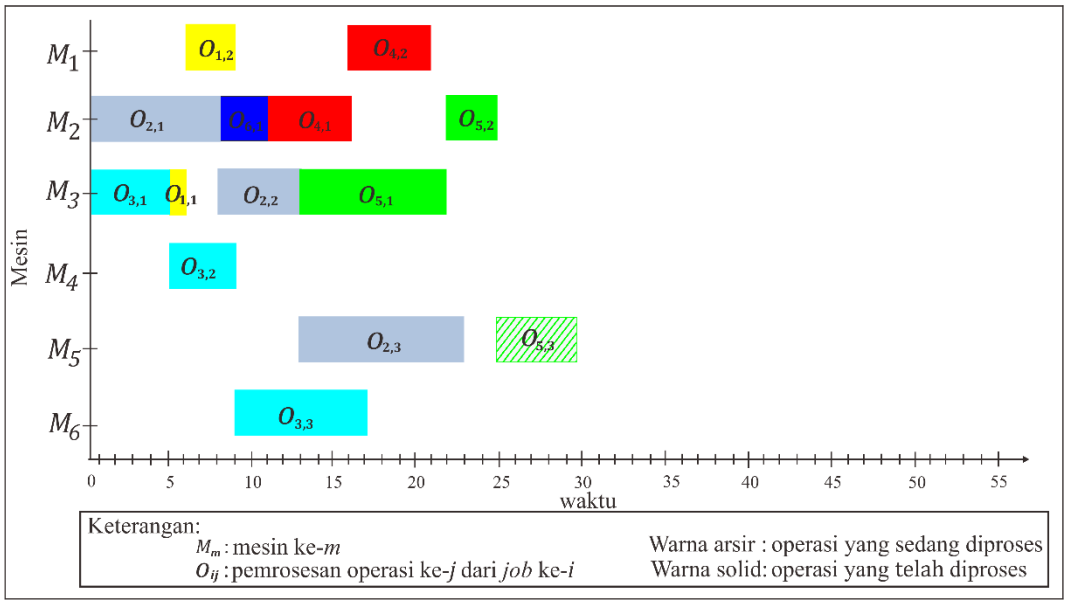


Gambar 3.20 Operasi ke-3 dari *job* 2 dikerjakan oleh mesin 5 selama 10 satuan waktu.

Gen ke-14 = $O_{5,3}$

Gen ke-14 merupakan $O_{5,3}$ yaitu operasi ke-3 dari *job* 5. Jika dikorespondensikan berdasarkan matrik mesin maka $O_{5,3} = M_5$, yang berarti operasi ke-3 dari *job* 5 akan diselesaikan oleh mesin 5. Jika dikorespondensikan berdasarkan matrik waktu maka $O_{5,3} = 5$, yang berarti operasi ke-3 dari *job* 5 akan diselesaikan selama 5 satuan waktu. Mesin 5 akan bekerja menyelesaikan operasi ke-3 dari *job* 5 setelah operasi ke-2 dari *job* 5 selesai dikerjakan oleh mesin 2 di waktu ke-25. Waktu awal mesin 5 bekerja menyelesaikan operasi ke-3 dari *job* 5 adalah di waktu ke-25 dan selesai di waktu ke- 30.

Operasi ke-3 dari *job* 5 dikerjakan oleh mesin 5 selama 5 satuan waktu dapat dilihat seperti dalam *Gantt chart* berikut.

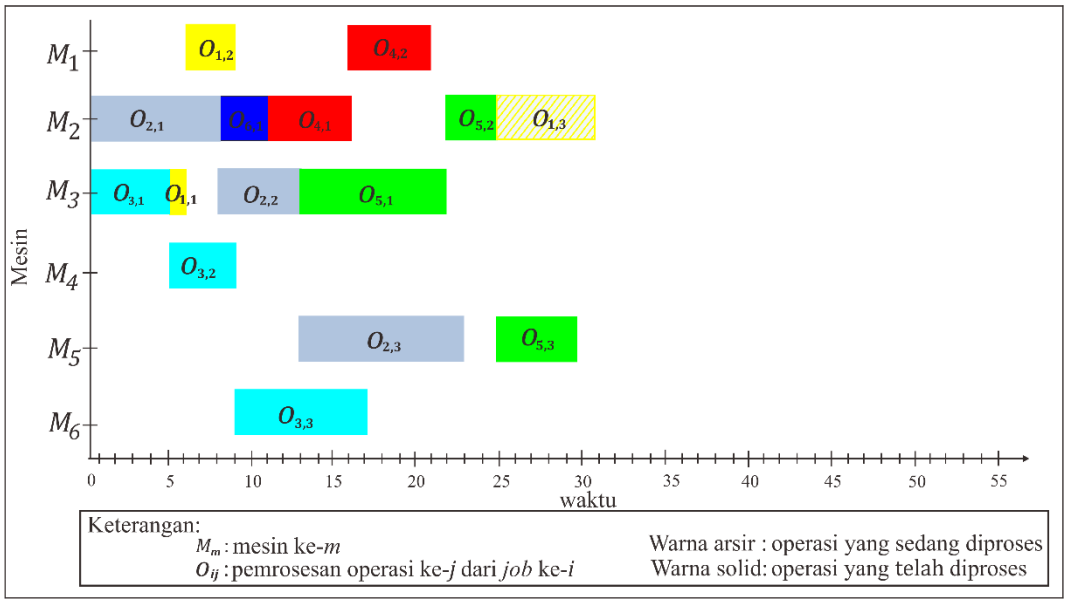


Gambar 3.21 Operasi ke-3 dari *job* 5 dikerjakan oleh mesin 5 selama 5 satuan waktu.

Gen ke-15 = $O_{1,3}$

Gen ke-15 merupakan operasi ke-3 dari *job* 3. Jika dikorespondensikan berdasarkan matrik mesin maka $O_{1,3} = M_2$, yang berarti operasi ke-3 dari *job* 1 akan diselesaikan oleh mesin 2. Jika dikorespondensikan berdasarkan matrik waktu maka $O_{1,3} = 6$, yang berarti operasi ke-3 dari *job* 1 akan diselesaikan selama 6 satuan waktu. Mesin 2 akan bekerja menyelesaikan operasi ke-3 dari *job* 1 setelah operasi ke-2 dari *job* 1 selesai dikerjakan oleh mesin 1 di waktu ke-9. Mesin 2 masih bekerja menyelesaikan $O_{6,1}$, $O_{4,1}$ dan $J_{5,2}$ setelah operasi ke-2 dari *job* 1 selesai dikerjakan oleh mesin 1 sehingga mesin 2 akan menegerjakan operasi ke-3 dari *job* 3 setelah $O_{5,2}$ selesai di waktu ke-25. Waktu awal mesin 2 bekerja menyelesaikan operasi ke-3 dari *job* 1 adalah di waktu ke-25 dan selesai di waktu ke- 31.

Operasi ke-3 dari *job* 1 dikerjakan oleh mesin 2 selama 6 satuan waktu dapat dilihat seperti dalam *Gantt chart* berikut.

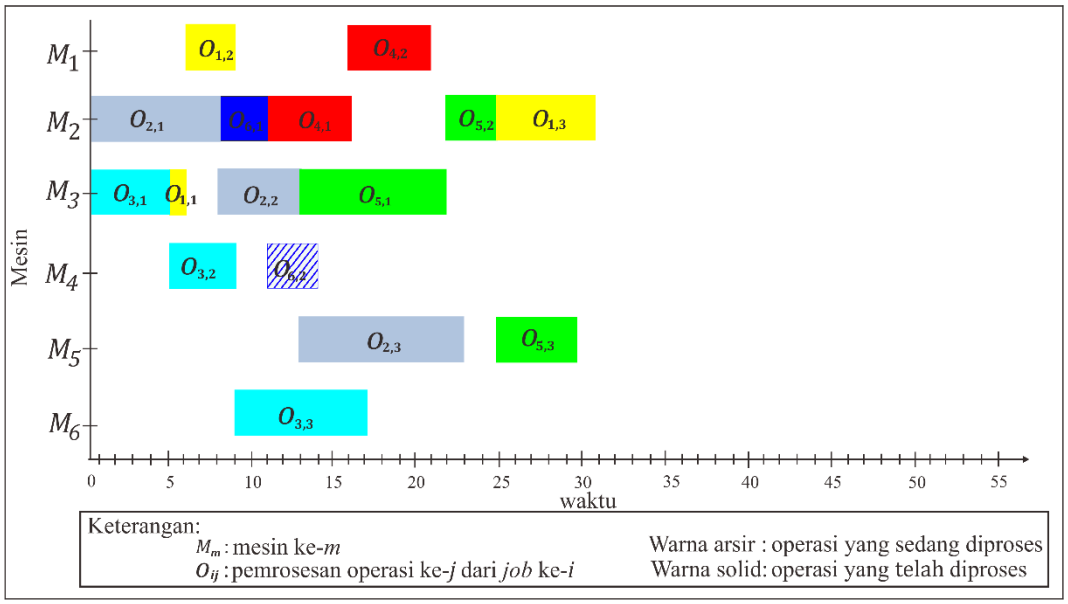


Gambar 3.22 Operasi ke-3 dari *job* 1 dikerjakan oleh mesin 2 selama 6 satuan waktu.

Gen ke-16 = $O_{6,2}$

Gen ke-16 merupakan operasi ke-2 dari *job* 6. Jika dikorespondensikan berdasarkan matrik mesin maka $O_{6,2} = M_4$, yang berarti operasi ke-2 dari *job* 6 akan diselesaikan oleh mesin 4. Jika dikorespondensikan berdasarkan matrik waktu maka $O_{6,2} = 3$, yang berarti operasi ke-2 dari *job* 6 akan diselesaikan selama 3 satuan waktu. Mesin 4 akan bekerja menyelesaikan operasi ke-2 dari *job* 6 setelah operasi ke-1 dari *job* 6 selesai dikerjakan oleh mesin 2 di waktu ke-11. Waktu awal mesin 4 bekerja menyelesaikan operasi ke-2 dari *job* 6 adalah di waktu ke-11 dan selesai di waktu ke- 14.

Operasi ke-2 dari *job* 6 dikerjakan oleh mesin 4 selama 3 satuan waktu dapat dilihat seperti dalam *Gantt chart* berikut.

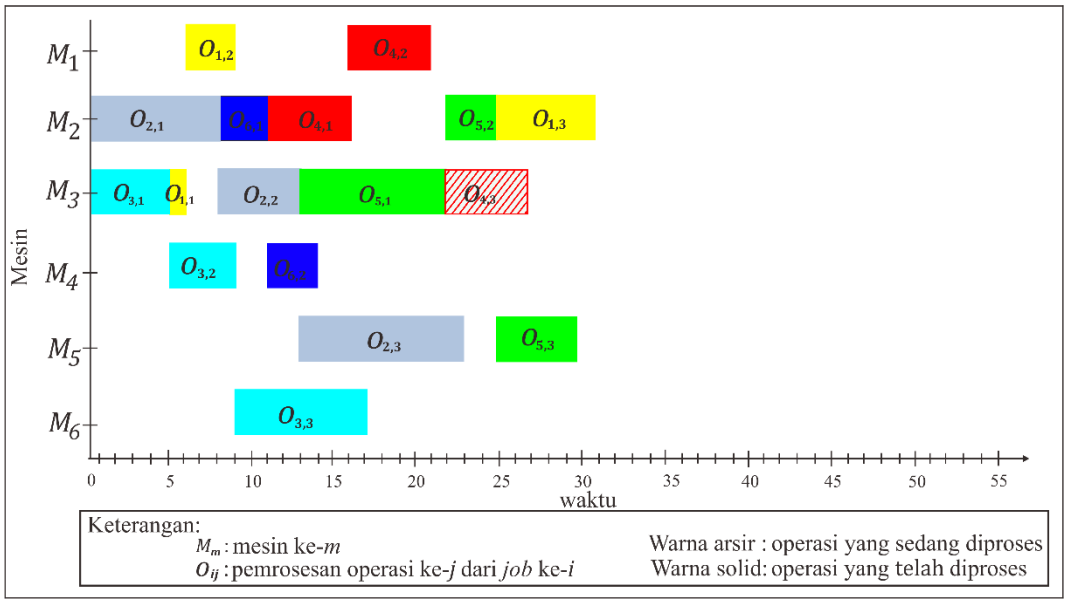


Gambar 3.23 Operasi ke-2 dari *job* 6 dikerjakan oleh mesin 4 selama 3 satuan waktu.

Gen ke-17 = $O_{4,3}$

Gen ke-17 merupakan $O_{4,3}$ yaitu operasi ke-3 dari *job* 4. Jika dikorespondensikan berdasarkan matrik mesin maka $O_{4,3} = M_3$, yang berarti operasi ke-3 dari *job* 4 akan diselesaikan oleh mesin 3. Jika dikorespondensikan berdasarkan matrik waktu maka $O_{4,3} = 5$, yang berarti operasi ke-3 dari *job* 4 akan diselesaikan selama 5 satuan waktu. Mesin 3 akan bekerja menyelesaikan operasi ke-3 dari *job* 4 setelah operasi ke-2 dari *job* 4 selesai dikerjakan oleh mesin 1. Mesin 3 selesai mengerjakan operasi ke-1 dari *job* 5 di waktu 22 sehingga *job* selanjutnya yang akan dikerjakan yaitu operasi ke-3 dari *job* 4 bisa dimulai untuk dikerjakan di waktu 22 . Waktu awal mesin 3 bekerja menyelesaikan operasi ke-2 dari *job* 4 adalah di waktu ke-22 dan selesai di waktu ke- 27.

Operasi ke-3 dari *job* 4 dikerjakan oleh mesin 3 selama 5 satuan waktu dapat dilihat seperti dalam *Gantt chart* berikut.

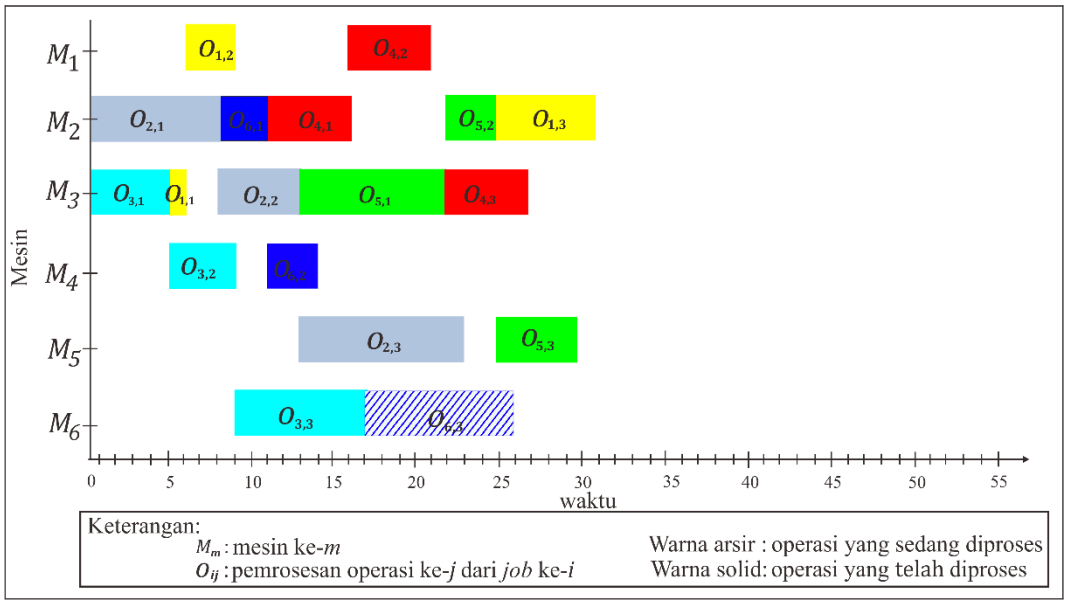


Gambar 3.24 Operasi ke-3 dari *job* 4 dikerjakan oleh mesin 3 selama 5 satuan waktu.

Gen ke-18 = $O_{6,3}$

Gen ke-18 merupakan operasi ke-3 dari *job* 6. Jika dikorespondensikan berdasarkan matrik mesin maka $O_{6,3} = M_6$, yang berarti operasi ke-3 dari *job* 6 akan diselesaikan oleh mesin 6. Jika dikorespondensikan berdasarkan matrik waktu maka $O_{6,3} = 9$, yang berarti operasi ke-3 dari *job* 6 akan diselesaikan selama 9 satuan waktu. Mesin 6 akan bekerja menyelesaikan operasi ke-3 dari *job* 6 setelah operasi ke-2 dari *job* 6 selesai dikerjakan oleh mesin 4. Mesin 3 selesai mengerjakan operasi ke-3 dari *job* 3 di waktu 17. sehingga *job* selanjutnya yang akan dikerjakan yaitu operasi ke-3 dari *job* 6 bisa dimulai untuk dikerjakan di waktu 17 . Waktu awal mesin 6 bekerja menyelesaikan operasi ke-3 dari *job* 6 adalah di waktu ke-17 dan selesai di waktu ke- 26.

Operasi ke-3 dari *job* 6 dikerjakan oleh mesin 6 selama 9 satuan waktu dapat dilihat seperti dalam *Gantt chart* berikut.

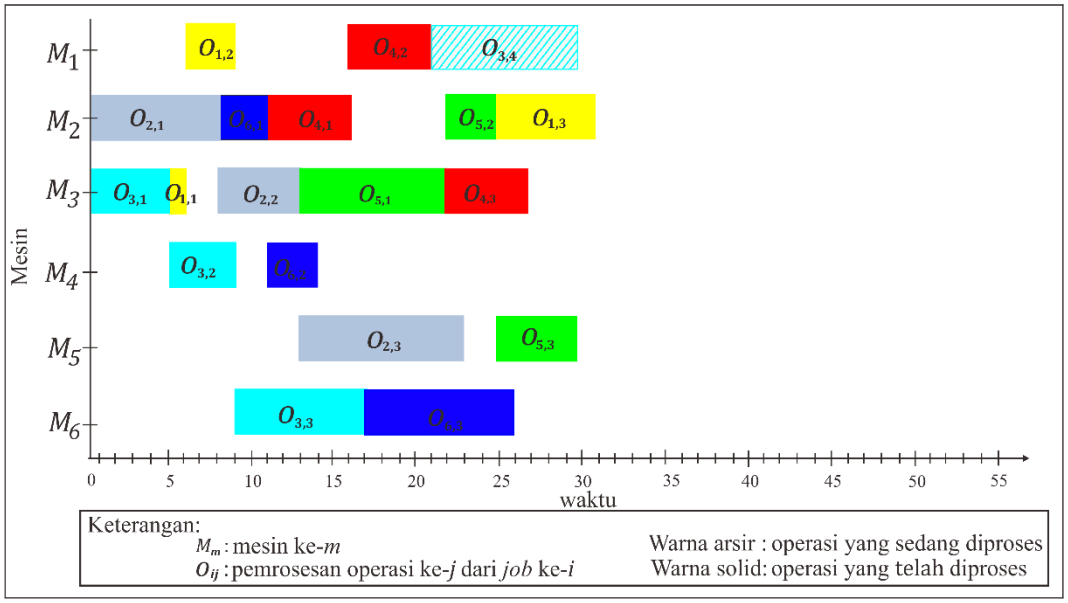


Gambar 3.25 Operasi ke-3 dari *job* 6 dikerjakan oleh mesin 6 selama 9 satuan waktu.

Gen ke-19 = $O_{3,4}$

Gen ke-19 merupakan operasi ke-4 dari *job* 3. Jika dikorespondensikan berdasarkan matrik mesin maka $O_{3,4} = M_1$, yang berarti operasi ke-3 dari *job* 4 akan diselesaikan oleh mesin 1. Jika dikorespondensikan berdasarkan matrik waktu maka $O_{3,4} = 9$, yang berarti operasi ke-3 dari *job* 4 akan diselesaikan selama 9 satuan waktu. Mesin 1 akan bekerja menyelesaikan operasi ke-4 dari *job* 3 setelah operasi ke-3 dari *job* 4 selesai dikerjakan oleh mesin 6. Mesin 1 selesai mengerjakan operasi ke-2 dari *job* 4 di waktu 21. sehingga *job* selanjutnya yang akan dikerjakan yaitu operasi ke-4 dari *job* 3 bisa dimulai untuk dikerjakan di waktu 21 . Waktu awal mesin 1 bekerja menyelesaikan operasi ke-4 dari *job* 3 adalah di waktu ke-21 dan selesai di waktu ke- 30.

Operasi ke-4 dari *job* 3 dikerjakan oleh mesin 1 selama 9 satuan waktu dapat dilihat seperti dalam *Gantt chart* berikut.



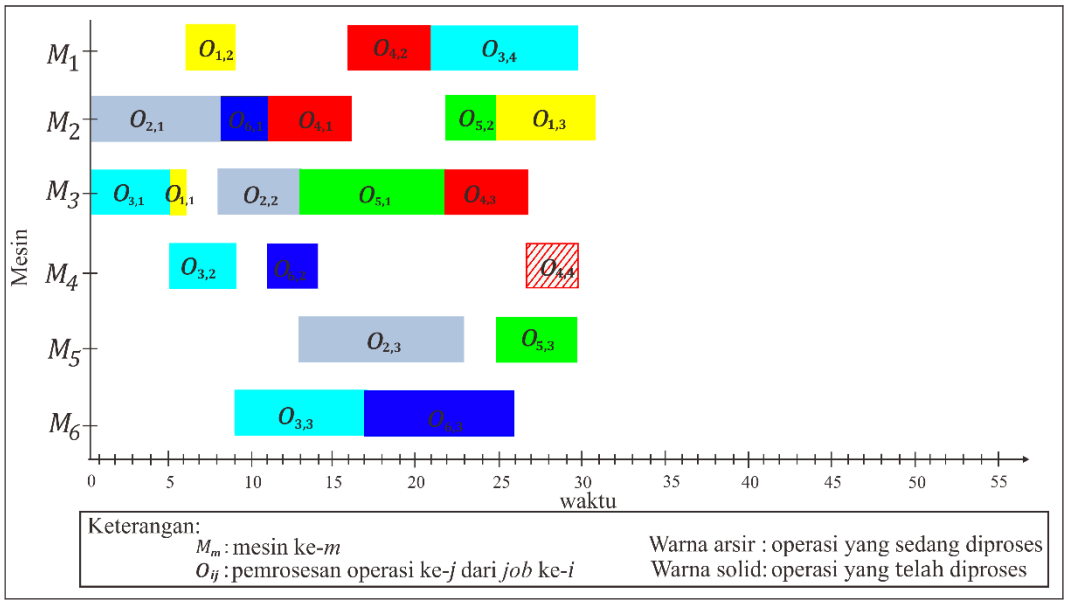
Gambar 3.26 Operasi ke-4 dari *job* 3 dikerjakan oleh mesin 1 selama 9 satuan waktu.

Gen ke-20 = $O_{4,4}$

Gen ke-20 merupakan operasi ke-4 dari *job* 4. Jika dikorespondensikan berdasarkan matrik mesin maka $O_{4,4} = M_4$, yang berarti operasi ke-4 dari *job* 4 akan diselesaikan oleh mesin 4. Jika dikorespondensikan berdasarkan matrik waktu maka $O_{4,4} = 3$, yang berarti operasi ke-4 dari *job* 4 akan diselesaikan selama 3 satuan waktu. Mesin 4 akan bekerja menyelesaikan operasi ke-4 dari *job* 4 setelah operasi ke-3 dari *job* 4 selesai dikerjakan oleh mesin 3 di waktu ke-27. Mesin 4 selesai mengerjakan operasi ke-2 dari *job* 6 di waktu 14. sehingga *job* selanjutnya bisa dikerjakan di waktu ke-14, akan tetapi belum ada *job* yang ada untuk diproses sehingga mesin 4 dalam keadaan menunggu sampai di waktu ke-27, operasi ke-4

dari *job* 4 siap untuk dikerjakan oleh mesin 4. Waktu awal mesin 4 bekerja menyelesaikan operasi ke-4 dari *job* 4 adalah di waktu ke-27 dan selesai di waktu ke- 30.

Operasi ke-4 dari *job* 4 dikerjakan oleh mesin 4 selama 3 satuan waktu dapat dilihat seperti dalam *Gantt chart* berikut.



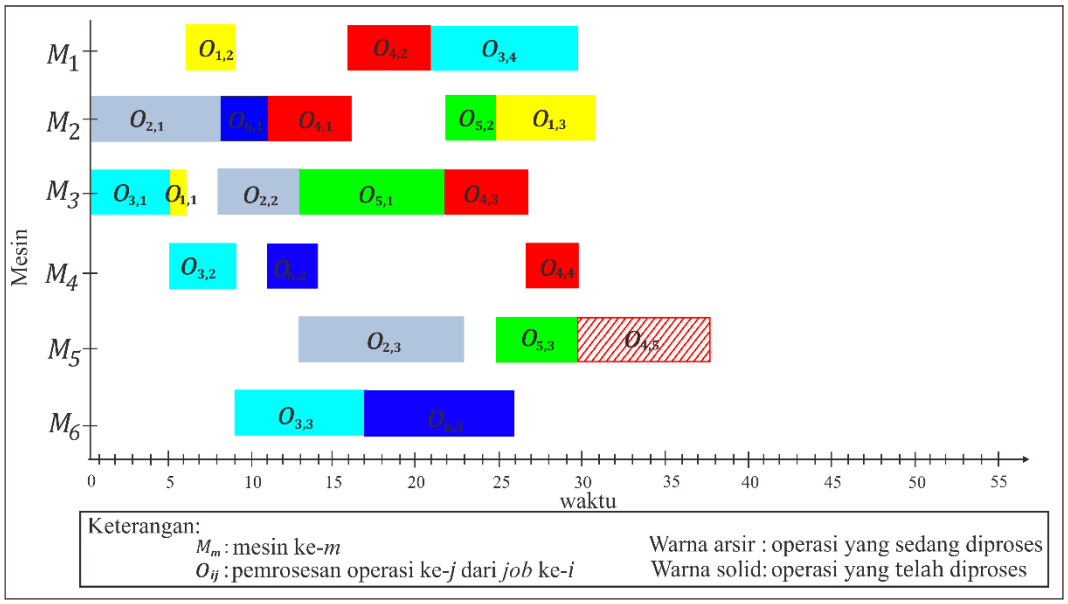
Gambar 3.27 Operasi ke-4 dari *job* 4 dikerjakan oleh mesin 4 selama 3 satuan waktu.

Gen ke-21 = $O_{4,5}$

Gen ke-21 merupakan operasi ke-5 dari *job* 4. Jika dikorespondensikan berdasarkan matrik mesin maka $O_{4,5} = M_5$, yang berarti operasi ke-5 dari *job* 5 akan diselesaikan oleh mesin 5. Jika dikorespondensikan berdasarkan matrik waktu maka $O_{4,5} = 8$, yang berarti operasi ke-5 dari *job* 4 akan diselesaikan selama 8 satuan waktu. Mesin 4 akan bekerja menyelesaikan operasi ke-5 dari *job* 4 setelah operasi ke-4 dari *job* 4 selesai dikerjakan oleh mesin 4 di waktu ke-30. Mesin 5

selesai mengerjakan operasi ke-3 dari *job* 5 di waktu 30 sehingga *job* selanjutnya bisa dikerjakan di waktu ke-30. operasi ke-5 dari *job* 4 datang siap untuk dikerjakan oleh mesin 5 di waktu ke-30 sehingga waktu awal mesin 4 bekerja menyelesaikan operasi ke-5 dari *job* 4 adalah di waktu ke-30 dan selesai di waktu ke- 38.

Operasi ke-5 dari *job* 4 dikerjakan oleh mesin 5 selama 8 satuan waktu dapat dilihat seperti dalam *Gantt chart* berikut.



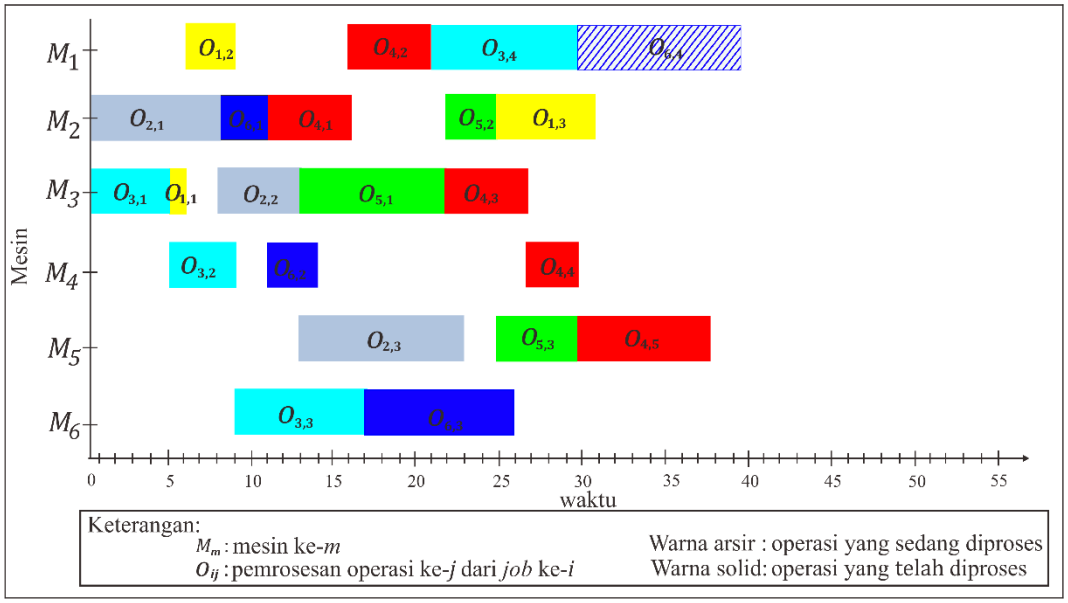
Gambar 3.28 Operasi ke-5 dari *job* 4 dikerjakan oleh mesin 5 selama 8 satuan waktu.

Gen ke-22 = $O_{6,4}$

Gen ke-22 merupakan operasi ke-4 dari *job* 6. Jika dikorespondensikan berdasarkan matrik mesin maka $O_{6,4} = M_1$, yang berarti operasi ke-4 dari *job* 6 akan diselesaikan oleh mesin 1. Jika dikorespondensikan berdasarkan matrik waktu maka $O_{6,4} = 10$, yang berarti operasi ke-4 dari *job* 6 akan diselesaikan selama 10 satuan waktu. Mesin 1 akan bekerja menyelesaikan operasi ke-4 dari *job* 6 setelah operasi ke-3 dari *job* 6 selesai dikerjakan oleh mesin 6. Mesin 1 selesai

mengerjakan operasi ke-4 dari *job* 3 di waktu 30 sehingga *job* selanjutnya bisa dikerjakan di waktu ke-30. operasi ke-4 dari *job* 6 akan dikerjakan oleh mesin 1 di waktu ke-30, sehingga waktu awal mesin 1 bekerja menyelesaikan operasi ke-4 dari *job* 6 adalah di waktu ke-30 dan selesai di waktu ke- 40.

Operasi ke-4 dari *job* 6 dikerjakan oleh mesin 1 selama 10 satuan waktu dapat dilihat seperti dalam *Gantt chart* berikut.



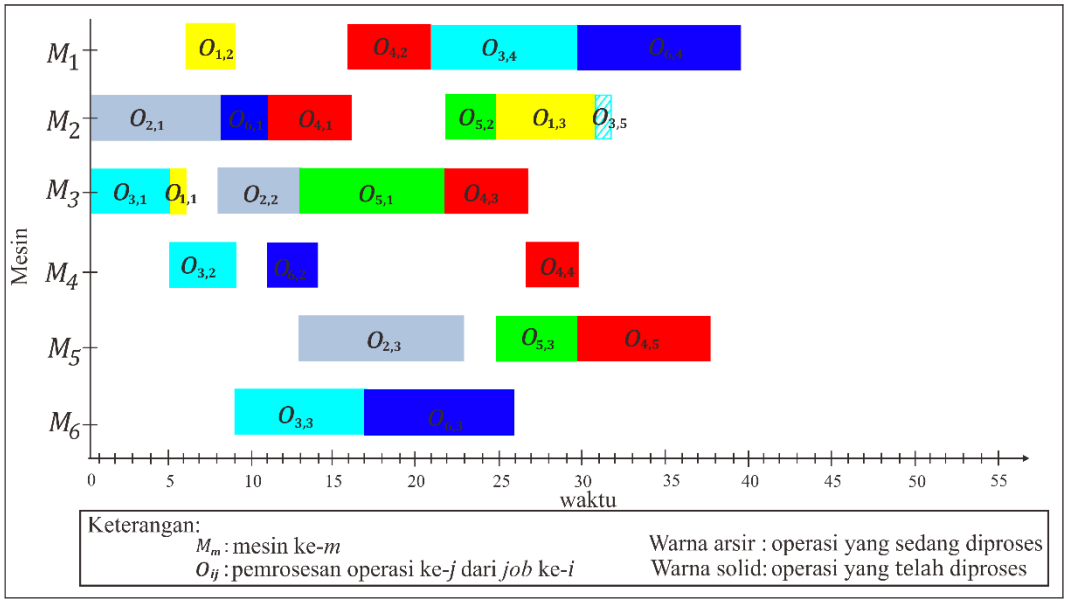
Gambar 3.29 Operasi ke-4 dari *job* 6 dikerjakan oleh mesin 1 selama 10 satuan waktu.

Gen ke-23 = $O_{3,5}$

Gen ke-23 merupakan operasi ke-5 dari *job* 3. Jika dikorespondensikan berdasarkan matrik mesin maka $O_{3,5} = M_2$, yang berarti operasi ke-3 dari *job* 5 akan diselesaikan oleh mesin 2. Jika dikorespondensikan berdasarkan matrik waktu maka $O_{3,5} = 1$, yang berarti operasi ke-5 dari *job* 3 akan diselesaikan selama 1 satuan waktu. Mesin 2 akan bekerja menyelesaikan operasi ke-5 dari *job* 3 setelah

operasi ke-4 dari *job* 3 selesai dikerjakan oleh mesin 1. Mesin 1 selesai mengerjakan operasi ke-4 dari *job* 3 di waktu 30, sedangkan mesin 2 selesai mengerjakan operasi ke-3 dari *job* 1 adalah diwaktu ke-31 sehingga operasi ke-5 dari *job* 3 bisa dikerjakan di mesin 2 di waktu ke-31. waktu awal mesin 2 bekerja menyelesaikan operasi ke-5 dari *job* 3 adalah di waktu ke-31 dan selesai di waktu ke- 32.

Operasi ke-5 dari *job* 3 dikerjakan oleh mesin 2 selama 1 satuan waktu dapat dilihat seperti dalam *Gantt chart* berikut.



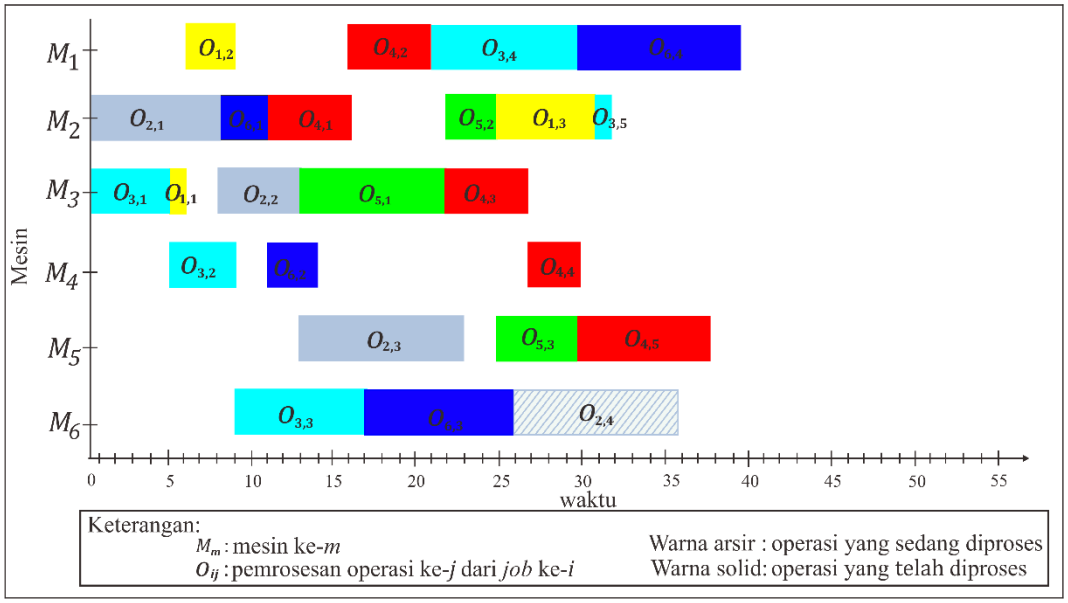
Gambar 3.30 Operasi ke-5 dari *job* 3 dikerjakan oleh mesin 2 selama 1 satuan waktu.

Gen ke-24 = $O_{2,4}$

Gen ke-24 merupakan operasi ke-4 dari *job* 2. Jika dikorespondensikan berdasarkan matrik mesin maka $J_{2,4} = M_6$, yang berarti operasi ke-4 dari *job* 2 akan diselesaikan oleh mesin 6. Jika dikorespondensikan berdasarkan matrik waktu maka $O_{2,4} = 10$, yang berarti operasi ke-4 dari *job* 2 akan diselesaikan selama 10

satuan waktu. Mesin 6 akan bekerja menyelesaikan operasi ke-4 dari *job* 2 setelah operasi ke-3 dari *job* 2 selesai dikerjakan oleh mesin 5. Mesin 5 selesai mengerjakan operasi ke-3 dari *job* 2 di waktu 23, sedangkan mesin 6 selesai mengerjakan operasi ke-3 dari *job* 6 diwaktu ke-26 sehingga operasi ke-4 dari *job* 2 baru bisa dikerjakan di mesin 6 di waktu ke-26. waktu awal mesin 6 bekerja menyelesaikan operasi ke-4 dari *job* 2 adalah di waktu ke-26 dan selesai di waktu ke- 36.

Operasi ke-4 dari *job* 2 dikerjakan oleh mesin 6 selama 10 satuan waktu dapat dilihat seperti dalam *Gantt chart* berikut.



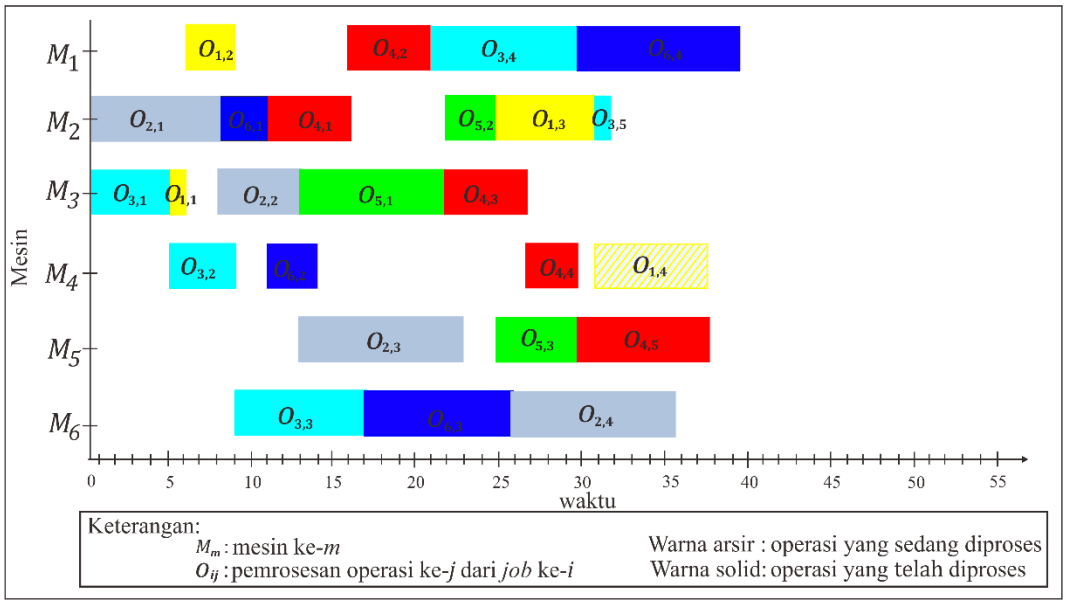
Gambar 3.31 Operasi ke-4 dari *job* 2 dikerjakan oleh mesin 6 selama 10 satuan waktu.

Gen ke-25 = $O_{1,4}$

Gen ke-25 merupakan operasi ke-4 dari *job* 1. Jika dikorespondensikan berdasarkan matrik mesin maka $O_{1,4} = M_4$, yang berarti operasi ke-4 dari *job* 1 akan diselesaikan oleh mesin 4. Jika dikorespondensikan berdasarkan matrik waktu

maka $O_{1,4}$ = 7 , yang berarti operasi ke-4 dari *job* 1 akan diselesaikan selama 7 satuan waktu. Mesin 4 akan bekerja menyelesaikan operasi ke-4 dari *job* 1 setelah operasi ke-3 dari *job* 1 selesai dikerjakan oleh mesin 2. Mesin 4 selesai mengerjakan operasi ke-4 dari *job* 4 di waktu 30, sedangkan mesin 2 selesai mengerjakan operasi ke-3 dari *job* 1 diwaktu ke-31 sehingga mesin ke-4 menunggu sampai operasi ke-4 dari *job* 1 datang untuk bisa dikerjakan yaitu di waktu ke-31. waktu awal mesin 4 bekerja menyelesaikan operasi ke-4 dari *job* 1 adalah di waktu ke-31 dan selesai di waktu ke- 38.

Operasi ke-4 dari *job* 1 dikerjakan oleh mesin 4 selama 7 satuan waktu dapat dilihat seperti dalam *Gantt chart* berikut.



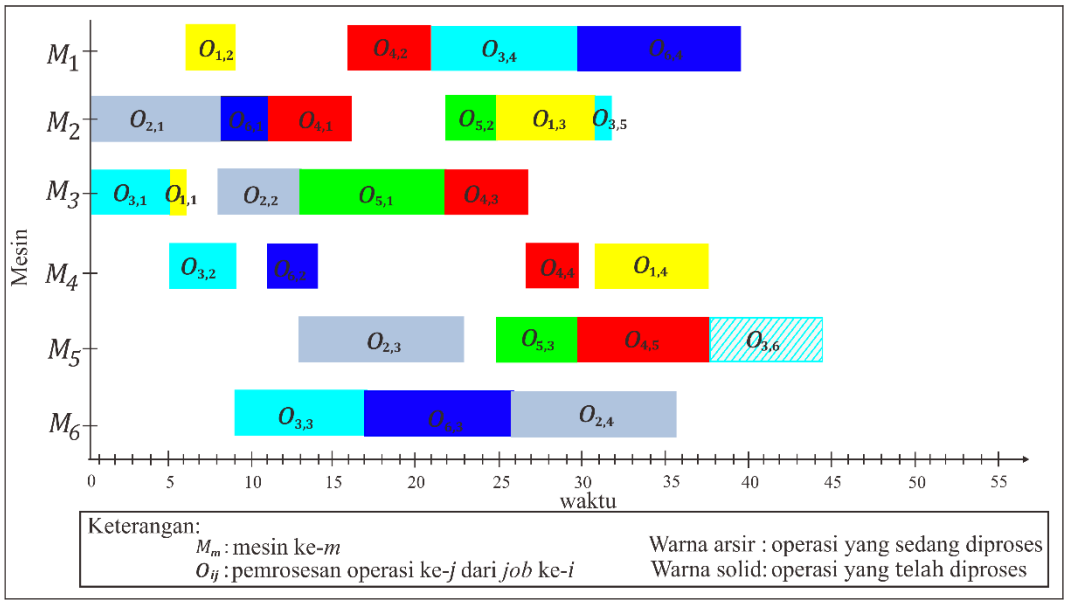
Gambar 3.32 Operasi ke-4 dari *job* 1 dikerjakan oleh mesin 4 selama 7 satuan waktu.

Gen ke-26 = $O_{3,6}$

Gen ke-26 merupakan operasi ke-6 dari *job* 3. Jika dikorespondensikan berdasarkan matrik mesin maka $O_{3,6}$ = M_5, yang berarti operasi ke-6 dari *job* 3

akan diselesaikan oleh mesin 5. Jika dikorespondensikan berdasarkan matrik waktu maka $O_{3,,6}$ = 7 , yang berarti operasi ke-6 dari *job* 3 akan diselesaikan selama 7 satuan waktu. Mesin 3 akan bekerja menyelesaikan operasi ke-6 dari *job* 3 setelah operasi ke-5 dari *job* 6 selesai dikerjakan oleh mesin 2. Mesin 5 selesai mengerjakan operasi ke-5 dari *job* 4 di waktu 38, sehingga baru bisa memproses operasi ke-6 dari *job* 3 di waktu ke-38. waktu awal mesin 5 bekerja menyelesaikan operasi ke-6 dari *job* 3 adalah di waktu ke-38 dan selesai di waktu ke- 45.

Operasi ke-6 dari *job* 3 dikerjakan oleh mesin 5 selama 7 satuan waktu dapat dilihat seperti dalam *Gantt chart* berikut.



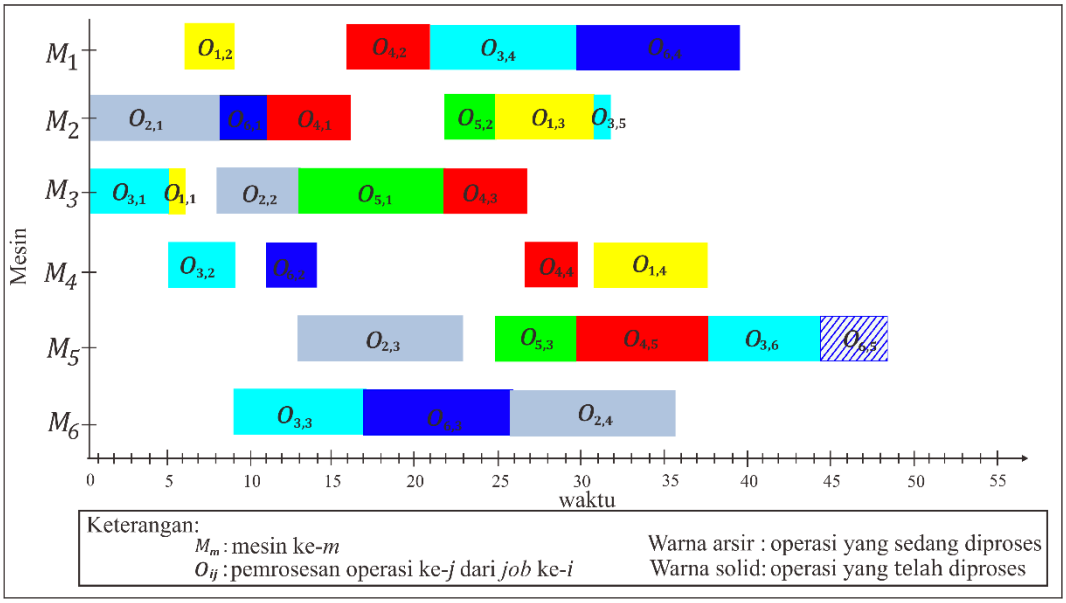
Gambar 3.33 Operasi ke-6 dari *job* 3 dikerjakan oleh mesin 5 selama 7 satuan waktu.

Gen ke-27 = $O_{6,5}$

Gen ke-27 merupakan operasi ke-5 dari *job* 6. Jika dikorespondensikan berdasarkan matrik mesin maka $O_{6,5}$ = M_5, yang berarti operasi ke-5 dari *job* 6 akan diselesaikan oleh mesin 5. Jika dikorespondensikan berdasarkan matrik waktu

maka $O_{6,5} = 4$, yang berarti operasi ke-5 dari *job* 6 akan diselesaikan selama 4 satuan waktu. Mesin 5 akan bekerja menyelesaikan operasi ke-5 dari *job* 6 setelah operasi ke-4 dari *job* 6 selesai dikerjakan oleh mesin 1. Mesin 1 selesai mengerjakan operasi ke-4 dari *job* 6 di waktu ke-38, sedangkan mesin 5 selesai mengerjakan operasi ke-6 dari *job* 3 diwaktu ke-45 sehingga mesin 5 baru bisa memproses operasi ke-5 dari *job* 6 di waktu ke-45. waktu awal mesin 5 bekerja menyelesaikan operasi ke-5 dari *job* 6 adalah di waktu ke-45 dan selesai di waktu ke- 49.

Operasi ke-5 dari *job* 6 dikerjakan oleh mesin 5 selama 4 satuan waktu dapat dilihat seperti dalam *Gantt chart* berikut.



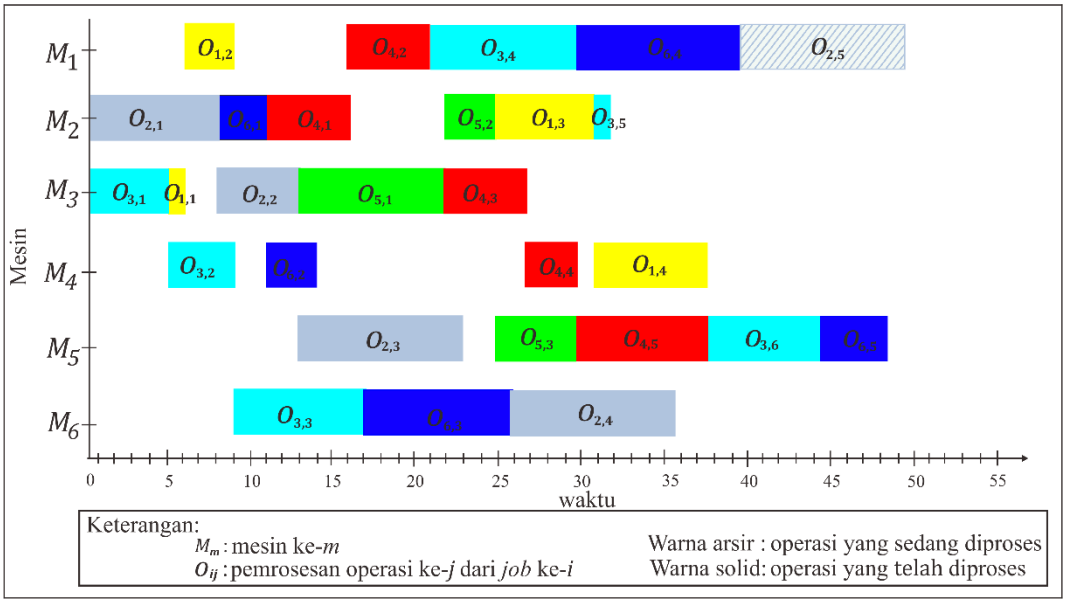
Gambar 3.34 Operasi ke-5 dari *job* 6 dikerjakan oleh mesin 5 selama 4 satuan waktu.

Gen ke-28 = $O_{2,5}$

Gen ke-28 merupakan operasi ke-5 dari *job* 2. Jika dikorespondensikan berdasarkan matrik mesin maka $O_{2,5} = M_1$, yang berarti operasi ke-5 dari *job* 2

akan diselesaikan oleh mesin 1. Jika dikorespondensikan berdasarkan matrik waktu maka $O_{2,5}$ = 10 , yang berarti operasi ke-5 dari *job* 2 akan diselesaikan selama 10 satuan waktu. Mesin 1 akan bekerja menyelesaikan operasi ke-5 dari *job* 2 setelah operasi ke-4 dari *job* 2 selesai dikerjakan oleh mesin 6. Mesin 6 selesai mengerjakan operasi ke-4 dari *job* 2 di waktu ke-38, sehingga mesin ke-1 bisa mengerjakan operasi ke-5 dari *job* 2 di waktu ke-38. waktu awal mesin 1 bekerja menyelesaikan operasi ke-5 dari *job* 2 adalah di waktu ke-38 dan selesai di waktu ke- 48.

Operasi ke-5 dari *job* 2 dikerjakan oleh mesin 1 selama 10 satuan waktu dapat dilihat seperti dalam *Gantt chart* berikut.



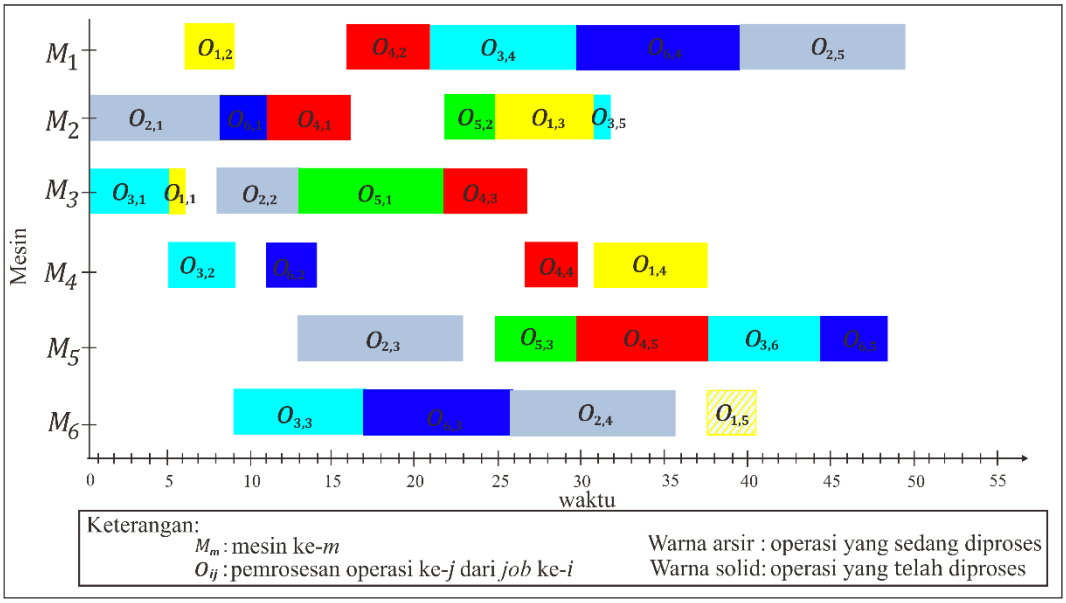
Gambar 3.35 Operasi ke-5 dari *job* 2 dikerjakan oleh mesin 1 selama 10 satuan waktu.

Gen ke-29 = $O_{1,5}$

Gen ke-29 merupakan operasi ke-5 dari *job* 1. Jika dikorespondensikan berdasarkan matrik mesin maka $O_{1,5}$ = M_6, yang berarti operasi ke-5 dari *job* 1 akan diselesaikan oleh mesin 6. Jika dikorespondensikan berdasarkan matrik waktu

maka $O_{1,5}$ = 3 , yang berarti operasi ke-5 dari *job* 1 akan diselesaikan selama 3 satuan waktu. Mesin 6 akan bekerja menyelesaikan operasi ke-5 dari *job* 1 setelah operasi ke-4 dari *job* 1 selesai dikerjakan oleh mesin 4. Mesin 6 selesai mengerjakan operasi ke-4 dari *job* 2 di waktu ke-38,sedangkan mesin 4 selesai mengerjakan operasi ke-4 dari *job* 1 diwaktu ke-37, sehingga mesin ke-6 harus menunggu operasi ke-4 dari *job* 1 selesai yaitu di waktu ke-38. Dengan demikian waktu awal mesin 6 bekerja menyelesaikan operasi ke-5 dari *job* 1 adalah di waktu ke-38 dan selesai di waktu ke- 41.

Operasi ke-5 dari *job* 1 dikerjakan oleh mesin 6 selama 3 satuan waktu dapat dilihat seperti dalam *Gantt chart* berikut.



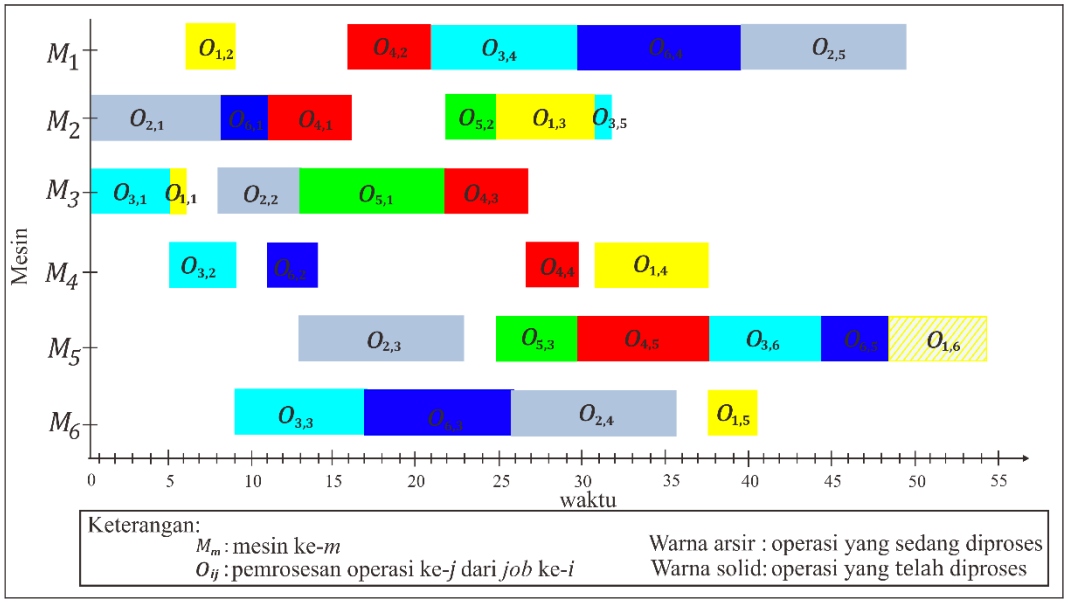
Gambar 3.36 Operasi ke-5 dari *job* 1 dikerjakan oleh mesin 6 selama 3 satuan waktu.

Gen ke-30 = $O_{1,6}$

Gen ke-30 merupakan operasi ke-6 dari *job* 1. Jika dikorespondensikan berdasarkan matrik mesin maka $O_{1,6} = M_5$, yang berarti operasi ke-6 dari *job* 1 akan diselesaikan oleh mesin 5. Jika dikorespondensikan berdasarkan matrik waktu

maka $O_{1,6}$ = 6 , yang berarti operasi ke-6 dari *job* 1 akan diselesaikan selama 6 satuan waktu. Mesin 5 akan bekerja menyelesaikan operasi ke-6 dari *job* 1 setelah operasi ke-5 dari *job* 1 selesai dikerjakan oleh mesin 6. Mesin 5 selesai mengerjakan operasi ke-5 dari *job* 6 di waktu ke-49, sehingga mesin ke-5 baru bisa memproses operasi ke-6 dari *job* 1 di waktu ke-49. Dengan demikian waktu awal mesin 5 bekerja menyelesaikan operasi ke-6 dari *job* 1 adalah di waktu ke-49 dan selesai di waktu ke- 55.

Operasi ke-6 dari *job* 1 dikerjakan oleh mesin 5 selama 6 satuan waktu dapat dilihat seperti dalam *Gantt chart* berikut.



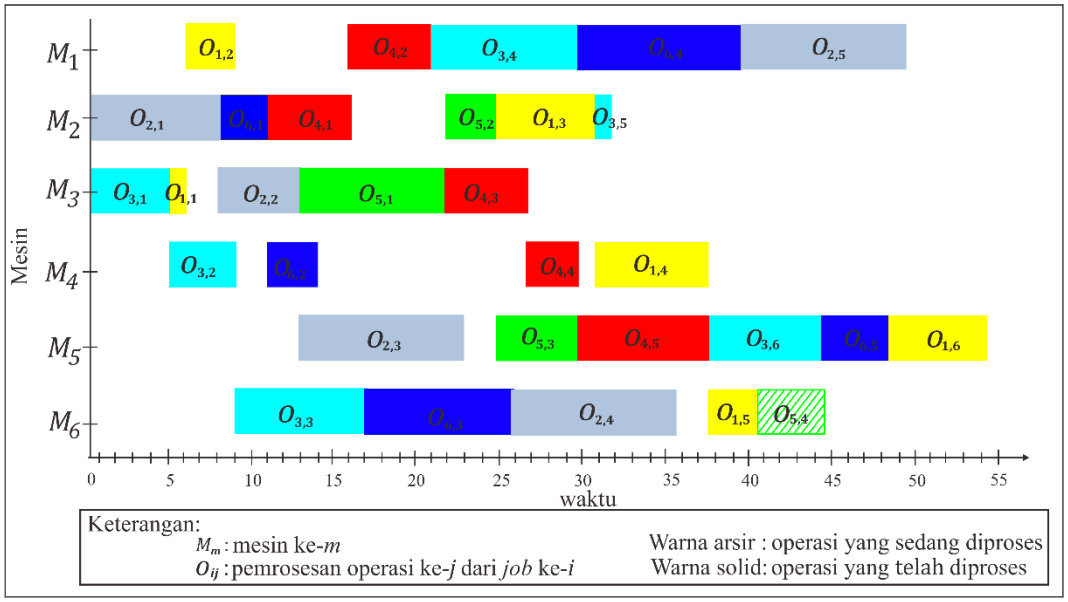
Gambar 3.37 Operasi ke-6 dari *job* 1 dikerjakan oleh mesin 5 selama 6 satuan waktu.

Gen ke-31 = $O_{5,4}$

Gen ke-31 merupakan operasi ke-4 dari *job* 5. Jika dikorespondensikan berdasarkan matrik mesin maka $O_{5,4} = M_6$, yang berarti operasi ke-4 dari *job* 5 akan diselesaikan oleh mesin 6. Jika dikorespondensikan berdasarkan matrik waktu

maka $O_{5,4}$ = 4 , yang berarti operasi ke-4 dari *job* 5 akan diselesaikan selama 4 satuan waktu. Mesin 6 akan bekerja menyelesaikan operasi ke-4 dari *job* 5 setelah operasi ke-3 dari *job* 5 selesai dikerjakan oleh mesin 5. Mesin 6 selesai mengerjakan operasi ke-5 dari *job* 1 di waktu ke-41, sehingga mesin 6 baru bisa memproses operasi ke-4 dari *job* 5 di waktu ke-41. Dengan demikian waktu awal mesin 5 bekerja menyelesaikan operasi ke-4 dari *job* 5 adalah di waktu ke-41 dan selesai di waktu ke- 45.

Operasi ke-4 dari *job* 5 dikerjakan oleh mesin 6 selama 4 satuan waktu dapat dilihat seperti dalam *Gantt chart* berikut.



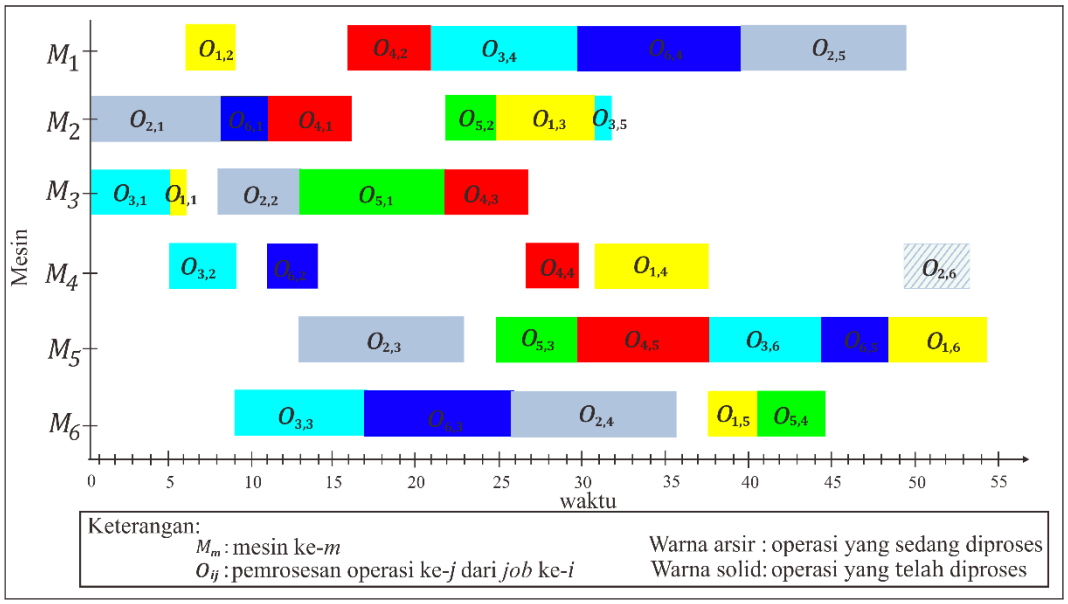
Gambar 3.38 Operasi ke-4 dari *job* 5 dikerjakan oleh mesin 6 selama 4 satuan waktu.

Gen ke-32 = $O_{2,6}$

Gen ke-32 yaitu $O_{2,6}$ yang merupakan operasi ke-6 dari *job* 2. Jika dikorespondensikan berdasarkan matrik mesin maka $O_{2,6}$ = M_4 , yang berarti operasi ke-6 dari *job* 2 akan diselesaikan oleh mesin 4. Jika dikorespondensikan

berdasarkan matrik waktu maka $J_{2,6} = 4$, yang berarti operasi ke-6 dari *job* 2 akan diselesaikan selama 4 satuan waktu. Mesin 4 akan bekerja menyelesaikan operasi ke-6 dari *job* 2 setelah operasi ke-5 dari *job* 2 selesai dikerjakan oleh mesin 1, sedangkan Mesin 4 selesai mengerjakan operasi ke-4 dari *job* 1 di waktu ke-37 sehingga mesin 4 harus menunggu operasi ke-5 dari *job* 2 selesai dikerjakan mesin 1 di waktu ke-48. Sehingga mesin 4 baru bisa memproses operasi ke-6 dari *job* 2 di waktu ke-48. Dengan demikian waktu awal mesin 4 bekerja menyelesaikan operasi ke-6 dari *job* 2 adalah di waktu ke-48 dan selesai di waktu ke- 52.

Operasi ke-6 dari *job* 2 dikerjakan oleh mesin 4 selama 4 satuan waktu dapat dilihat seperti dalam *Gantt chart* berikut.



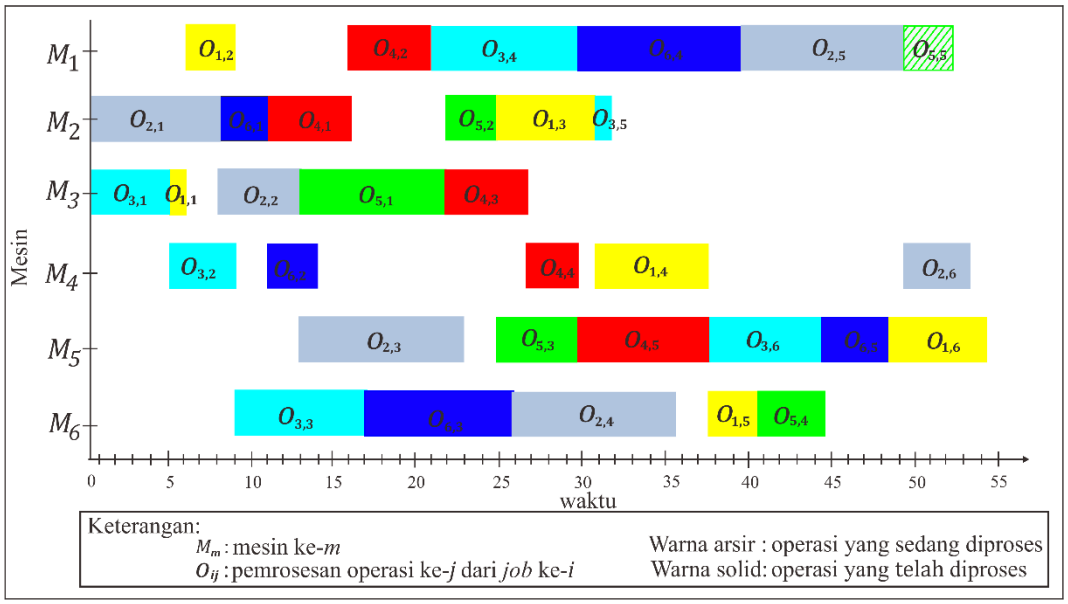
Gambar 3.39 Operasi ke-6 dari *job* 2 dikerjakan oleh mesin 4 selama 4 satuan waktu.

Gen ke-33 = $O_{5,5}$

Gen ke-33 merupakan $O_{5,5}$ yaitu operasi ke-5 dari *job* 5. Jika dikorespondensikan berdasarkan matrik mesin maka $O_{5,5} = M_1$, yang berarti operasi ke-5 dari *job* 5

akan diselesaikan oleh mesin 1. Jika dikorespondensikan berdasarkan matrik waktu maka $O_{5,5} = 3$, yang berarti operasi ke-5 dari *job* 5 akan diselesaikan selama 3 satuan waktu. Mesin 1 akan bekerja menyelesaikan operasi ke-5 dari *job* 5 setelah operasi ke-4 dari *job* 5 selesai dikerjakan oleh mesin 6 dan setelah mesin 1 selesai menyelesaikan operasike-5 dari *job* 2 yaitu diwaktu ke-48, sehingga mesin 1 baru bisa memproses operasi ke-5 dari *job* 5 di waktu ke-48. Dengan demikian waktu awal mesin 1 bekerja menyelesaikan operasi ke-5 dari *job* 5 adalah di waktu ke-48 dan selesai di waktu ke- 51.

Operasi ke-5 dari *job* 5 dikerjakan oleh mesin 1 selama 3 satuan waktu dapat dilihat seperti dalam *Gantt chart* berikut.



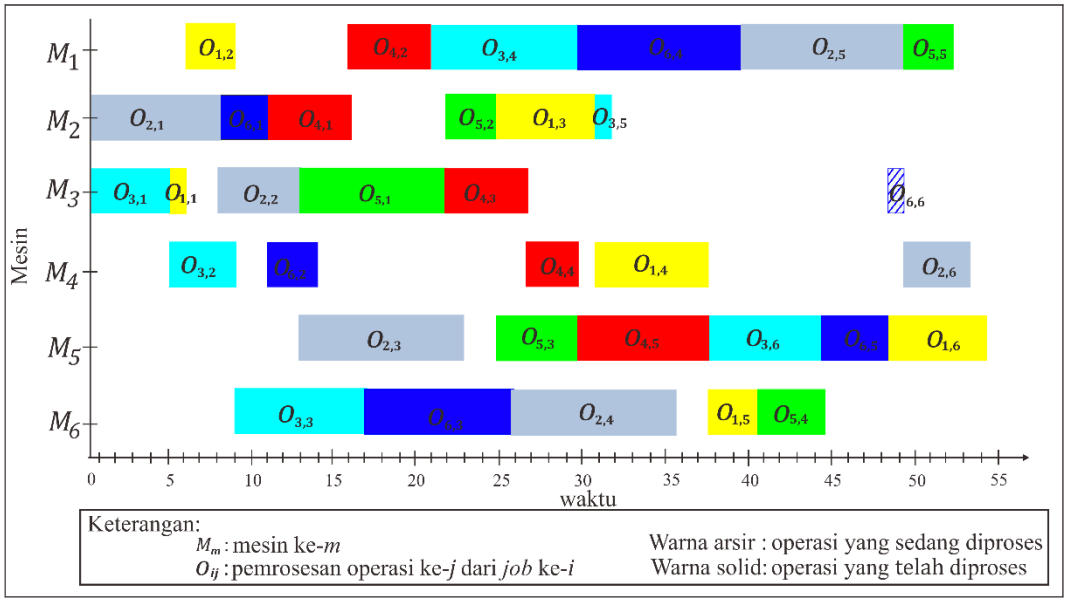
Gambar 3.40 Operasi ke-5 dari *job* 5 dikerjakan oleh mesin 1 selama 3 satuan waktu.

Gen ke-34 = $O_{6,6}$

Gen ke-34 merupakan $O_{6,6}$ yaitu operasi ke-6 dari *job* 6. Jika dikorespondensikan berdasarkan matrik mesin maka $O_{6,6} = M_3$, yang berarti operasi ke-6 dari *job* 6

akan diselesaikan oleh mesin 3. Jika dikorespondensikan berdasarkan matrik waktu maka $O_{6,6} = 1$, yang berarti operasi ke-6 dari *job* 6 akan diselesaikan selama 1 satuan waktu. Mesin 3 akan bekerja menyelesaikan operasi ke-6 dari *job* 6 setelah operasi ke-5 dari *job* 6 selesai dikerjakan oleh mesin 5 yaitu diwaktu ke-49 sehingga mesin 3 baru akan memproses operasi ke-6 dari *job* 6 di waktu ke-49. Dengan demikian waktu awal mesin 3 bekerja menyelesaikan operasi ke-6 dari *job* 6 adalah di waktu ke-49 dan selesai di waktu ke- 50.

Operasi ke-6 dari *job* 6 dikerjakan oleh mesin 3 selama 1 satuan waktu dapat dilihat seperti dalam *Gantt chart* berikut.



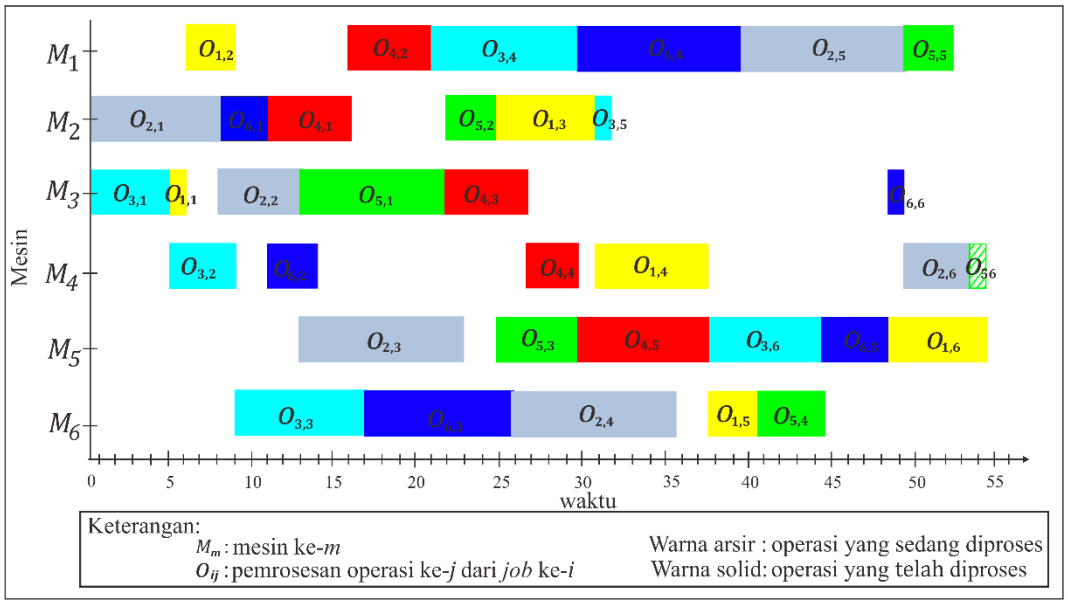
Gambar 3.41 Operasi ke-6 dari *job* 6 dikerjakan oleh mesin 3 selama 1 satuan waktu.

Gen ke-35 = $O_{5,6}$

Gen ke-35 merupakan $O_{5,6}$ yaitu operasi ke-6 dari *job* 5. Jika dikorespondensikan berdasarkan matrik mesin maka $O_{5,6} = M_4$, yang berarti operasi ke-6 dari *job* 5 akan diselesaikan oleh mesin 4. Jika dikorespondensikan berdasarkan matrik waktu

maka $O_{5,6}$ = 1 , yang berarti operasi ke-6 dari *job* 5 akan diselesaikan selama 1 satuan waktu. Mesin 4 akan bekerja menyelesaikan operasi ke-6 dari *job* 5 setelah operasi ke-5 dari *job* 5 selesai dikerjakan oleh mesin 1 dan operasi ke-6 dari *job* 2 telah selesai dikerjakan di mesin 4. operasi ke-5 dari *job* 5 selesai dikerjakan Mesin 1 di waktu ke 51 sedangkan operasi ke-6 dari *job* 2 baru selesai di waktu ke-52 sehingga mesin 4 baru akan memproses operasi ke-6 dari *job* 6 di waktu ke-52. Dengan demikian waktu awal mesin 4 bekerja menyelesaikan operasi ke-6 dari *job* 5 adalah di waktu ke-52 dan selesai di waktu ke- 53.

Operasi ke-6 dari *job* 5 dikerjakan oleh mesin 4 selama 1 satuan waktu dapat dilihat seperti dalam *Gantt chart* berikut.

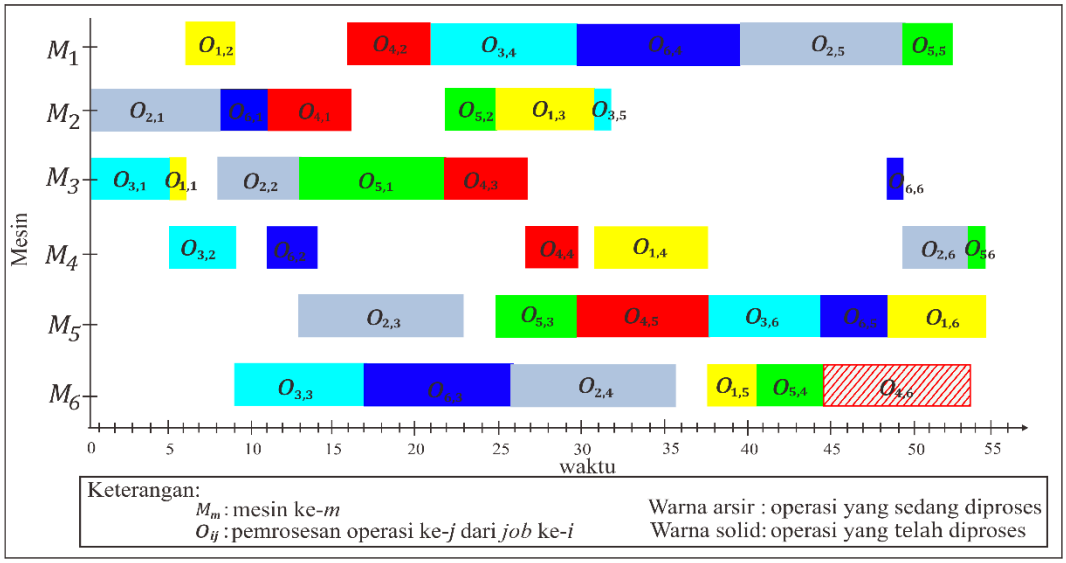


Gambar 3.42 Operasi ke-6 dari *job* 5 dikerjakan oleh mesin 4 selama 1 satuan waktu.

Gen ke-36 = $O_{4,6}$

Gen ke-36 merupakan $O_{4,6}$ dan merupakan gen terakhir dari kromosom. $O_{4,6}$ berarti yaitu operasi ke-6 dari *job* 4. Jika dikorespondensikan berdasarkan matrik

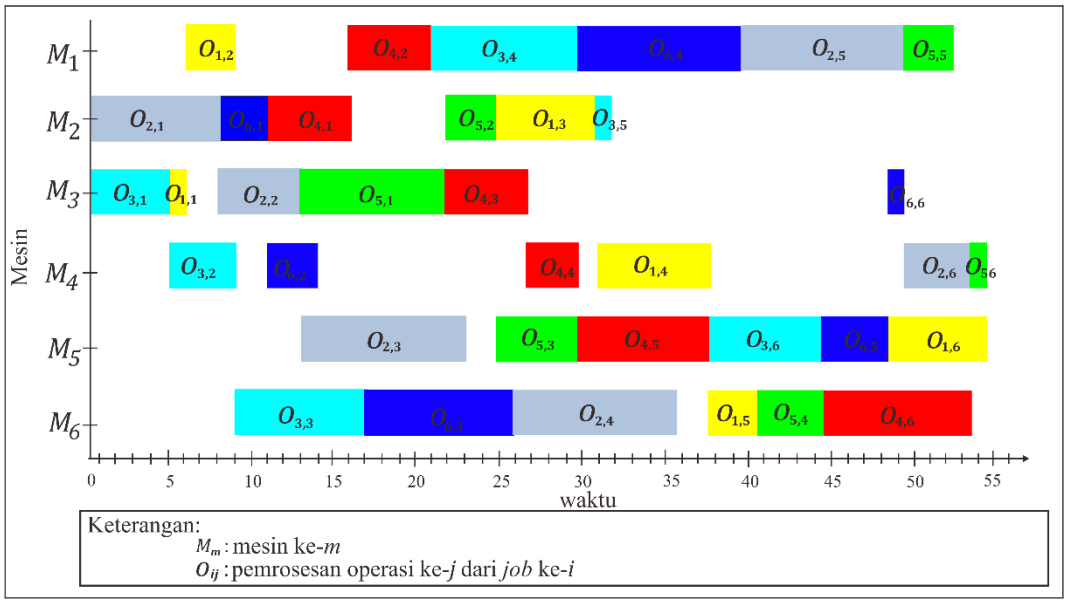
mesin maka $O_{4,6} = M_6$, yang berarti operasi ke-6 dari *job* 4 akan diselesaikan oleh mesin 6. Jika dikorespondensikan berdasarkan matrik waktu maka $O_{4,6} = 9$, yang berarti operasi ke-6 dari *job* 4 akan diselesaikan selama 9 satuan waktu. Mesin 6 akan bekerja menyelesaikan operasi ke-6 dari *job* 4 setelah operasi ke-5 dari *job* 4 selesai dikerjakan oleh mesin 5 dan operasi ke-4 dari *job* 5 telah selesai dikerjakan di mesin 6. operasi ke-5 dari *job* 4 selesai dikerjakan Mesin 5 di waktu ke 38 sedangkan operasi ke-4 dari *job* 5 baru selesai di waktu ke-45 sehingga mesin 6 baru akan memproses operasi ke-6 dari *job* 4 di waktu ke-45. Dengan demikian waktu awal mesin 6 bekerja menyelesaikan operasi ke-6 dari *job* 4 adalah di waktu ke-45 dan selesai di waktu ke- 54. Operasi ke-6 dari *job* 4 dikerjakan oleh mesin 6 selama 9 satuan waktu dapat dilihat seperti dalam *Gantt chart* berikut.



Gambar 3.43 Operasi ke-6 dari *job* 4 dikerjakan oleh mesin 6 selama 9 satuan waktu.

Setelah semua gen pada kromosom diproses bearti semua operasi dari semua *job* telah diproses oleh mesin dan waktu maksimum penyelesaian semua operasi *job* tersebut disebut *makespan* .pada proses dekode kromosom ini *makespan* yang

diperoleh adalah 55 satuan waktu. Hasil akhir proses dekode kromosom diperoleh jadwal seperti pada gambar berikut.



Gambar 3.44 Jadwal hasil dekode kromosom dari *problem* FT06.

6. Metode Seleksi

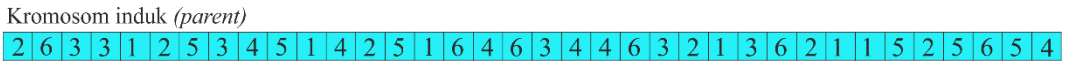
Metode seleksi yang diterapkan dalam penelitian ini adalah metode seleksi *Elitist.* Pada Metode seleksi ini setiap individu dihitung nilai *fitness* nya. Semua individu akan berkesempatan untuk berkompetensi untuk maju kegenerasi berikutnya, Individu yang memiliki nilai *fitness* tinggi akan terbawa ke generasi berikutnya untuk dilakukan proses *crossover* dan mutasi. Metode seleksi *Elitist* menjamin bahwa individu terbaik pasti maju kegenerasi berikutnya.

7. Metode pindah silang (*crossover*)

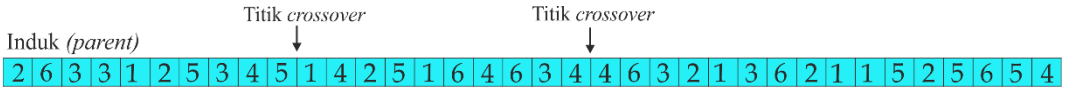
Untuk memperoleh hasil yang lebih bervariasi dan tidak terjebak pada nilai lokal optimum maka proses *Crossover* dan mutasi sangat diperlukan. Metode *crossover*

yang dipilih harus sesuai dengan jenis representasi kromosom yang digunakan. Pada penelitian ini skema pengkodean permutasi dengan perulangan (*permutation with repetition*) yang diperkenalkan oleh Bierwirth (1995). Pada penelitian ini operator rekombinasi yang digunakan yaitu metode *self crossover* (Hou, *et al*., 2011). Metode *crossover* ini mengadopsi cara reproduksi beberapa jenis spesies makhluk hidup yang dilakukan dengan dirinya sendiri tanpa melibatkan induk lain (*asexual reproduction*) (Sibly & Calow, 1982). Berikut prosedur metode *self crossover :*

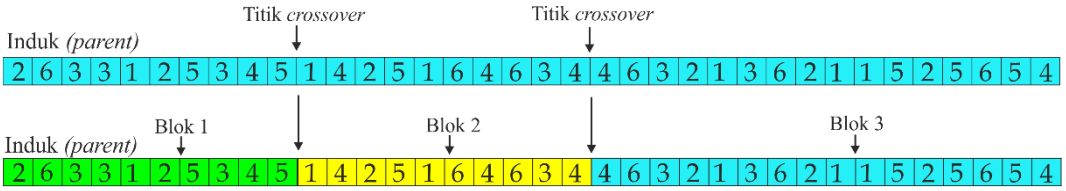
Step 1: Pilih satu kromosom tunggal dari populasi sebagai kromosom induk (*parent*).



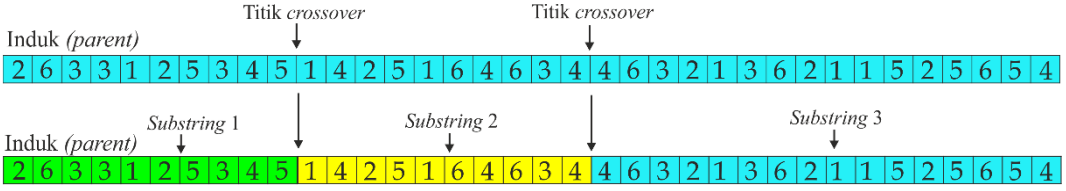
Step 2 : Tentukan 2 titik *crossover* secara acak pada kromosom induk



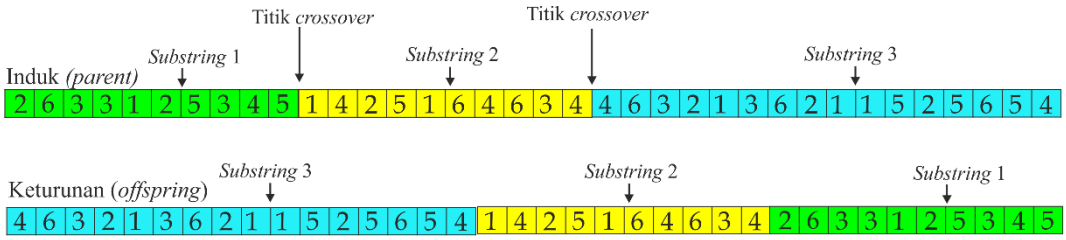
Setelah titik *crossover* di tentukan maka aka terbentuk blok diantara titik *crossover*



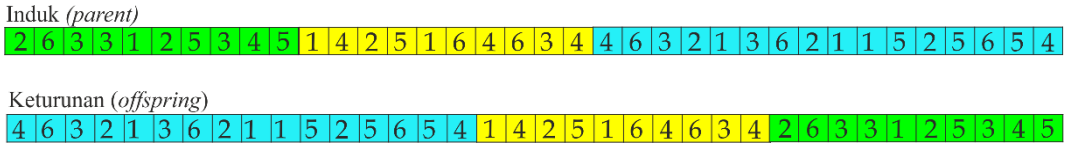
Step 3: pilih *substring* diantara 2 titik *crossover* pada kromosom induk untuk dilakukan proses *crossover*.



Step 4: Pindah silangkan elemen atau *substring* pada blok yang terletak diantara 2 titik *crossover* di induk kromosom ke kromosom *offspring*.



Step 5: setelah proses *crossover* selesai maka akan menghasilkan 2 individu yaitu *parent* dan *offspring.*



Step 6: Ulangi step 1 sampai step 4 untuk menghasilkan sejumlah *offspring*.

8. Mutasi

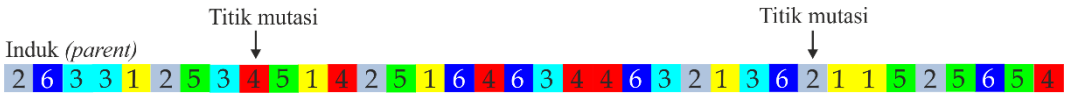
Metode mutasi yang akan digunakan dalam penelitian ini adalah metode petukaran (*swap mutation*). Proses ini dilakukan dengan cara mengambil dua titik pada kromosom secara acak dan selanjutnya menukarkan masing-masing gen pada kedua titik tersebut. Ilustrasi dapat dijelaskan seperti gambar berikut:

Gambar 3.45 Metode *swap mutation*

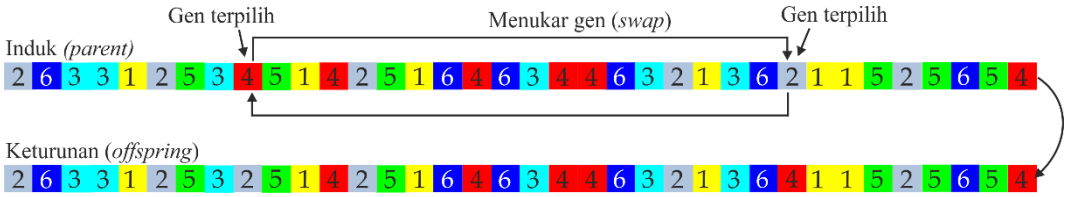
Secara rinci *swap mutation* dapat dijelaskan dengan prosedur berikut.

Step 1. Pilih kromosom dari populasi yang akan dilakukan proses mutasi.

Step 2. Tentukan titik mutasi pada kromosom terpilih. Titik mutasi yaitu pada gen ke-9 yang berelemen 4 dan gen ke- 27 yaitu berelemen 2



Step 3.. Pilih elemen pada titik mutasi untuk dipertukarkan. Proses ini ini disebut proses penukaran (*swap*) . Proses ini menukarkan elemen gen ke-9 yang bernilai 4 dengan gen ke-27 yang bernilai 2.



Step 4. Setelah proses penukaran elemen dilakukan maka akan terbentuk individu baru (*offspring*) yang mewarisi karakter kromosom induk.

Ulangi step 1 sampai step 3 untuk menghasilkan sejumlah *offspring*.

9. Kriteria pemberhentian (*stoping criteria*)

Kriteria pemberhentian yang dipilih pada penelitian ini adalah maksimum generasi (*max_gen*). Pada tahap iterasi GA telah mencapai generasi maksimum maka proses GA akan berhenti, jika belum tercapai generasi maksimum maka kembali pada langkah evaluasi fungsi *fitness*.

3.5 Pengujian Program

Pengujian program pada penelitian ini dilakukan dengan pengujian menggunakan data uji. Data uji terdiri dari data *test* problem yang merupakan *instance benchmarks* persoalan JSSP dan juga data nilai optimal (*Best Known Solution*)

untuk masing masing *instance benchmarks*. *Test problem* yang digunakan yaitu *instance benchmark* dari Fisher dan Thompson (1963) dan Lawrence (1984). Pengujian program dilakukan dengan berbagai variasi nilai parameters GA. Pengujian dijalankan pada Processor Intel(R) Core (TM) i3-2365M CPU @1.40GHz, RAM 2 GB

Setiap *instance benchmarks* memiliki ukuran dimensi yang merupakan informasi dari jumlah *Job* dan jumlah mesin. data *test problem* FT06, FT10, dan FT20 dibuat oleh Fisher dan Thompson (1963) sedangkan data *test* problem LA01-la40 dibuat oleh Lawrence (1984) . Semua data *test problem* tersebut tersedia di OR *Library* yang dapat diakses di halaman http://people.brunel.ac.uk/~mastjjb/jeb/orlib/files/*Job*shop1.txt. Pengujian yang digunakan dalam penelitian ini pada rentang ukuran *benchmark* sebesar (6 *job*×6 mesin) , (10 *job* ×10 mesin) , (15 *job*×10 mesin) , (20 *job*×5 mesin) , (20 *job*×10 mesin), (15 *job*×15 mesin) dan (30 *job*×10 mesin). Berikut ini data *test problem* yang akan dipakai untuk pengujian kinerja GA dalam penyelesaian persoalan JSSP.

Tabel 3.3 Data uji untuk persoalan JSSP

No	*Test problem* (*Instance*)*	Nama Alternatif*	Dimensi ($n \times m$)*	*Best Known Solution* (BKS)**
1	FT05	MT06	6×6	55
2	FT10	MT10	10×10	930
3	FT20	MT10	20×5	1165
4	LA01	F1	10×5	666
5	LA02	F2	10×5	655
6	LA03	F3	10×5	597
7	LA04	F4	10×5	590
8	LA05	F5	10×5	593

Sumber: *Beasley (1990) dan **Gonçalves *et al* (2005)

Tabel 3.3 (Lanjutan)

No	*Test problem* (*Instance*)*	Nama Alternatif*	Dimensi ($n \times m$) *	*Best Known Solution* (BKS)**
9	LA06	G1	15×5	926
10	LA07	G2	15×5	890
11	LA08	G3	15×5	863
12	LA09	G4	15×5	951
13	LA10	G5	15×5	958
14	LA11	H1	20×5	1222
15	LA12	H2	20×5	1039
16	LA13	H3	20×5	1150
17	LA14	H4	20×5	1292
18	LA15	H5	20×5	1207
19	LA16	A1	10×5	945
20	LA17	A2	10×10	784
21	LA18	A3	10×10	848
22	LA19	A4	10×10	842
23	LA20	A5	10×10	902
24	LA21	B1	15×10	1046
25	LA22	B2	15×10	927
26	LA23	B3	15×10	1032
27	LA24	B4	15×10	935
28	LA25	B5	15×10	977
29	LA26	C1	20×10	1218
30	LA27	C2	20×10	1235
31	LA28	C3	20×10	1216
32	LA29	C4	20×10	1157
33	LA30	C5	20×10	1355
34	LA31	D1	30×10	1784
35	LA32	D2	30×10	1850
36	LA33	D3	30×10	1719
37	LA34	D4	30×10	1721
38	LA35	D5	30×10	1888
39	LA36	I1	15×15	1268
40	LA37	I2	15×15	1397
41	LA38	I3	15×15	1196
42	LA39	I4	15×15	1233
43	LA40	I5	15×15	1222

Sumber: *Beasley (1990) dan **Gonçalves *et al* (2005)

3.6 *Numerical Experiment*

Hasil eksperimen yang telah diperoleh selanjutnya akan dibandingan dengan nilai optimal atau BKS (*Best Known Solution*) dan juga dengan beberapa metode metode yang pernah dipublikasikan. Metode metaheuristik yang akan dibandingkan pada penelitian ini adalah:

1. Hibridasi GA dengan *Local Search* atau *Hybrid Genetic Algorithm* (HGA)
    - El-Desoky *et al* (2016)
    - Gonçalves *et al* (2005)
2. *Artificial Immune System* (AIS)
    - Muhamad *et al* (2013)
3. *Harmony Bacterial Swarming Algorithm* (HBSA)
    - Shivakumar & Amudha (2012)
4. *Ant Colony Optimization* (ACO)
    - Flórez, et al (2013)
5. *Particle Swarm Optimization* (PSO)
    - Pongchairerks (2009)
6. *Multi-Objective Particle Swarm Optimization* (MOPSO)
    - Sha & Lin (2009)
7. *Hybrid Harmony Search Algorithm* (HHSA)
    - Piroozfard (2015)
8. *Tabu Search* (TS)
    - Nowicki & Smutnick (1996)

V. KESIMPULAN

5.1 Kesimpulan

Dari hasil percobaan dan pembahasan yang telah dilakukan dapat diambil kesimpulan sebagai berikut:

1. GA merupakan algoritma yang cukup efektif untuk digunakan dalam menyelesaikan persoalan JSSP, karena memberikan waktu komputasi yang cepat dengan hasil yang cukup optimal.
2. Metode GA efektif mencapai solusi optimal dengan nilai rata rata *relative error* tidak lebih dari 1.14% pada penyelesaian 43 *benchmark test problem* JSSP yaitu *problem* FT06-FT20 (Fisher & Thompson, 1963) dan *problem* LA01-LA40 (Lawrence, 1984).
3. Hasil perbandingan beberapa metode metaheuristik terhadap 28 data uji menunjukan bahwa metode GA lebih efektif dari metode *Artificial Immune System* (AIS) (Muhamad, *et al*., 2015), *Ant Colony Optimization* (ACO) (Flórez, *et al*., 2013), *Particle Swarm Optimization* **(**PSO) (Pongchairerks, 2009), *multi-objective Particle Swarm Optimization* (MOPSO) (Sha & Lin, 2009). Metode GA terkadang juga lebih unggul dibandingkan dengan metode hibridasi seperti metode *Hybrid Genetic Algorithm* (HGA) (El-Desoky , *et al*., 2016), *Hybrid Harmony Search Algorithm* (HHSA) (Piroozfard, *et al.*, 2015),

Harmony Bacterial Swarming Algorithm (HBSA) (Shivakumar & Amudha, 2012) meskipun GA efektif tetapi tidak lebih efektif dari *Tabu Search* (TS) (Nowicki & Smutnicki, 1996).

5.2 Saran

1. Kepada peneliti lain dapat menyelesaikan persoalan JSSP dengan metode Hibridasi GA (*Hybrid Genetic Algorithm*) dengan berbagai metode metaheuristik lain dengan mengambil keunggulan fitur masing masing metode agar hasil yang didapatkan diharapkan lebih optimal.
2. Kepada peneliti lain dapat mengevaluasi metode representasi kromosom dan evaluasi operator reproduksi dan operator rekombinasi GA untuk lebih memaksimalkan hasil optimasi.
3. Pada penelitian masa depan bisa menyelesaikan penjadwalan JSSP dengan fungsi multi tujuan (*multiple objective function*) seperti *mean flow time, mean completion time, maximum tardiness, maximum lateness* dan *due dates* sebagai kriteria kualitas penjadwalan.

DAFTAR PUSTAKA

Beasley, D., Bull, D. R. & Martin, . R. R., 1993. An Overview of Genetic Algorithms: Part 1, Fundamentals. *University Computing*, 15(2), pp. 58-69.

Adams, J., Balas, E. & Zawack, D., 1988 . The Shifting Bottleneck Procedure for Job Shop Scheduling. *Management Science*, 1 March, 34(3), pp. 391 - 401.

Akram, K., Kamal, K. & Zeb, A., 2016. Fast simulated annealing hybridized with quenching for solving job shop scheduling problem. *Journal of Applied Soft Computing*, 01 12, 49(C), p. 510–523.

Alharkan, I. M., 2005. *Algorithms for Sequencing and Scheduling.* Riyadh, Saudi Arabia: Industrial Engineering Department, King Saud University.

Antoniou, A. & Lu, W. S., 2007. *Practical Optimization Algorithms and Engineering Applications.* New York,USA: Springer Science+Business Media, LLC.

Applegate, D. & Cook, W., 1991. A Computational Study of the Job-Shop Scheduling Problem. *ORSA Journal on Computing*, 3(12), pp. 149-156.

Baker, K. R. & Trietsch, D., 2009. *Principles of Sequencing and Scheduling.* Hoboken, New Jersey, USA: John Wiley & Sons, Inc..

Baker, K. R., 1974. *Introduction to Sequencing and scheduling.* New York: John Wiley.

Balas , E., 1969. Machine sequencing via disjunctive graphs: an implicit enumeration algorithm. *Operations Research*, Volume 17, p. 941–957.

Baptiste, P., Flamini, M. & Sourd, F., 2006. Lagrangian bounds for just-in-time job-shop scheduling. *Computers & Operations Research*, 35(2008), p. 906–915.

Beasley, J. E., 1990. OR-Library: distributing test problems by electronic mail. *Journal of the Operational Research Society*, 41(11), pp. 1069-1072.

Beasley, . J. E., 2004. *Brunel University London.* [Online] Available at: http://people.brunel.ac.uk/~mastjjb/jeb/orlib/files/jobshop1.txt [Diakses 11 September 2017].

Bierwirth, C., 1995. A generalized permutation approach to job shop scheduling with genetic algorithms. *Operations-Research-Spektrum,* June, 17(2-3), p. 87–92.

Brucker, P., Burke, E. K. & Groenemeyer, S., 2012. A branch and bound algorithm for the cyclic job-shop problem with transportation. *Computers & Operations Research,* 14 April, Volume 39, p. 3200–3214.

Catanzaro, D., Gouveia, L. & Labbé, M., 2015. Improved integer linear programming formulations for the job Sequencing and tool Switching Problem. *European Journal of Operational Research,* Volume 244, p. 766–777.

Conway, R. W., W.L. Maxwell & L.W. Miller , 1967. *Theory of Scheduling.* Reading, MA: Addison-Wesley.

Darwin, C., 1859. *Origin of the species.*

Dell'Amico, M. & Trubian, M., 1993. Applying Tabu Search to the JobShop Scheduling Problem. *Annals of Operations Research,* Volume 41, pp. 231-252.

Demirkol, E., Mehta, S. & Uzsoy , R., 1998. Benchmarks for shop scheduling problems. *European Journal of Operational Research,* Volume 109, pp. 137- 141.

El-Desoky, I. M., El-Shorbagy, M. A., Nasr, S. M., Hendawy, Z. M., & Mousa, A. A. 2016. A Hybrid Genetic Algorithm for Job Shop Scheduling Problems. *International Journal of Advancement in Engineering, Technology and Computer Sciences*, 3(1), pp. 6-17.

Fang, H.-L., Ross, P. & Corne, D., 1993. *A Promising Genetic Algorithm Approach to Job-Shop Scheduling, Rescheduling,and Open-Shop Scheduling Problems.* San Mateo: Morgan Kaufmann, s.n., pp. 375-382.

Fera, M., Fruggiero, F., Lambiase, A., Martino, G., & Nenni, M. E. 2015. Production Scheduling Approaches for Operations Management. In: *Operations Management.* s.l.:InTech, pp. 113-139.

Fisher, H. & Thompson, G., 1963. Probabilistic learning combinations of local job-shop scheduling rules. Dalam: *Industrial Scheduling (In: Muth, J.F.; Thompson, G.L. (eds.)).* Englewood Cliffs: Prentice Hall.

Flórez, E., Gómez, W. & Bautista, M. L., 2013. *An Ant Colony Optimization Algorithm For Job Shop Scheduling Problem.* Argentine, ASAI, pp. 72-84.

Florian, M., Trepant, P. & McMahon, G., 1971. An Implicit Enumeration Algorithm for the Machine Sequencing Problem. *Management Science,* 17(12), pp. 782-792.

GanttHenry, L., 1916. *Work,Wages,andProfits.* 2nd ed. NewYork: Engineering Magazine Co..

Garey, M. R., Johnson, D. S. & Sethi , R., 1976. The Complexity of Flowshop and Jobshop Scheduling. *Mathematics of Operations Research,* May, 1(2), pp. 117-129.

Gen, M. & Cheng, R., 2000. *Genetic Algorithms and Engineering Optimization.* New York,USA: John Wiley and Sons.

Gen, M., Tsujimura, Y. & Kubota, E., 1994. Solving Job-Shop Scheduling Problems by Genetic Algorithm. *IEEE,* pp. 1577-1582.

Glover, F. & Greenberg, H. J., 1989. New approaches for heuristic search: A bilateral linkage with artificial intelligence. *European Journal of Operational Research,* May, Volume 39, pp. 119-130.

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

Gonçalves, J. F., Mendes, J. J. d. M. & Resende, M. G., 2005. A Hybrid Genetic Algorithm for the Job Shop Scheduling Problem. *European Journal of Operational Research,* Volume 167, p. 77–95.

Greenberg, H. J. & Glover, F., 1989. New approaches for heuristic search:A bilateral linkage with artificial intelligence. *European Journal of Operational Research,* Volume 39, pp. 119-130.

Gromicho, J. A., van Hoorn, J. J., Saldanha-da-Gama, F. & Timmer, G. T., 2012. Solving the job-shop scheduling problem optimally by dynamic programming. *Computers & Operations Research,* 8 March , Volume 39, p. 2968–2977.

Haupt, R. L. & Haupt, S. E., 2004. *Practical genetic algorithms.* 2nd ed. Hoboken, NJ: John Wiley & Sons, Inc.

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

Hou, S., Liu, Y., Wen, H. & Chen, Y., 2011. *A Self-Crossover Genetic Algorithm for Job Shop Scheduling Problem.* s.l., IEEE, pp. 549-554.

Jain, A. S. & Meeran, S., 1998. Deterministic job-shop scheduling: Past, present and future. *uropean Journal of Operation Research,* 113(1999), p. 390 – 434.

Kumar, R., 2012. Blending Roulette Wheel Selection & Rank Selection in Genetic Algorithms. *International Journal of Machine Learning and Computing,* 2(4), pp. 365-370.

Lageweg, B. J., Lenstra, J. K. & Rinnooy Kan, A., 1977. Jhop-Shop Scheduling by implicit enumeration. *Management Science,* 24(4), pp. 441-450.

Lawrence, S., 1984. *Resource constrained project scheduling: an experimental investigation of heuristic scheduling techniques (Supplement),* Pittsburgh, Pennsylvania: Graduate School of Industrial Administration, Carnegie-Mellon University.

Lenstra, J. K. & Rinnooy Kan, A., 1979. Computational complexity of discrete optimisation. *Annals of Discrete Mathematics,* Volume 4, pp. 121-140.

Mehmood, N., Umer, M., Ahmad, R. & Rafique, A. F., 2013. Optimization of Makespan and Mean Flow Time for Job Shop Scheduling Problem FT06 Using ACO. *Life Science Journal,* 10(4), pp. 477-484.

Mesghouni, K. & Hammadi, S., 2004. Evolutionary Algorithms For Job-Shop Scheduling. *Int. J. Appl. Math. Comput. Sci.,* 14(1), p. 91–103.

Michalewicz, z., 1996. *Genetic Algorithm + Data Structures = Evolution Programs.* 3rd ed. Berlin Heidelberg: Springer-Verlag.

Mitchell, M., 1995. Genetic Algorithms: An Overview. *Complexity,* 1(1), pp. 31-39.

Morton, T. E. & Pentico, D. W., 1993 . *Heuristic scheduling systems : with applications to production systems and project management.* New York, NY, United States of America : John Wiley & Sons, Inc. .

Muhamad, A. S., Deris, S. & Zakaria, Z., 2015. Minimize The Makespan For Job Shop Scheduling Problem Using Artificial Immune System Approach. *Journal of Theoretical and Applied Information Technology,* 80(1), pp. 141-146.

Nowicki, E. & Smutnicki, C., 1996. A Fast Taboo Search Algorithm for the Job-Shop Problem. *Institute for Operations Research and the Management Sciences,* 42(6), pp. 797-813.

Özgüven, C., Yavuz, Y. & Özbakır, L., 2012. Mixed integer goal programming models for the flexible job-shop scheduling problems with separable and non-separable sequence dependent setup times. *Applied Mathematical Modelling,* 36(2), p. 846–858.

Pinedo, M. L., 2008. *Scheduling: Theory, Algorithms, and Systems.* 3rd ed. New York: Springer Publishing Company, Incorporated.

Piroozfard, H., Wong, K. Y. & Asl, A. D., 2015. *A Hybrid Harmony Search Algorithm for the Job Shop Scheduling Problems.* s.l., IEEE, pp. 48-52.

Ploydanai, K. & Mungwattana, A., 2010. Algorithm for Solving Job Shop Scheduling Problem Based on machine availability constraint. *(IJCSE) International Journal on Computer Science and Engineering,* 02(05), pp. 1919-1925.

Pongchairerks, P., 2009. Particle swarm optimization algorithm applied to scheduling problems. *ScienceAsia,* 35(2009), p. 89–94.

Roshanaei , V., Balagh , A. K. G., Seyyed Esfahani , M. M. & Vahdani, B., 2010. A mixed-integer linear programming model along with an electromagnetism-like algorithm for scheduling job shop production system with sequence-dependent set-up times. *Int J Adv Manuf Technol,* Volume 47, p. 783–793.

Roy, B. & Sussman, B., 1964. *Les problèmes d'ordonnancement avec contraintes disjunctive,* Note D.S.,Paris: Technical Report 9,SEMA.

Sha, D. Y. & Lin, H. H., 2009. *A Multi-objective PSO for Job-shop Scheduling Problems.* s.l., IEEE, pp. 489-494.

Shivakumar, B. L. & Amudha, T., 2012. A Hybrid Bacterial Swarming Methodology for Job Shop Scheduling Environment. *Global Journal of Computer Science and Technology Hardware & Computation,* 12(10), pp. 7-16.

Sibly, R. & Calow, P., 1982. Asexual Reproduction in Protozoa and Invertebrates. *J. theor. Biol.,* 4 December, Volume 96, pp. 401- 424.

Sivanandam, S. & Deepa, S., 2008. *Introduction to Genetic Algorithm.* Berlin Heidelberg: Springer-Verlag.

Sotskov, Y. N. & Shakhlevich, N. V., 1995. NP-hardness of shop-scheduling problems with three jobs. *Discrete Applied Mathematics,* 59(1995), pp. 237-266.

Storer , R. H., Wu, . S. . D. & Vaccari, . R., 1992. New search spaces for sequencing instances with application to job shop. *Management Science,* 38(10), pp. 1495-1509 .

Syarif, A., 2014. *Algoritma Genetika, Teori dan Aplikasi.* 2nd ed. Ruko Jambusari 7A Yogyakarta 55283: Graha Ilmu.

Taillard, E., 1993. Benchmarks for basic scheduling problems. *Eur J Oper Res,* Volume 64, p. 278 – 285..

Thamilselvan, R. & Balasubramanie , P., 2012. Integrating Genetic Algorithm, Tabu Search and Simulated Annealing For Job Shop Scheduling Problem. *International Journal of Computer Applications ,* 48(5), pp. 42-54.

Uzorh, A. C. & Innocent, N., 2014. Solving Machine Shops Scheduling Problemsusing Priority Sequencing Rules Techniques. *The International Journal Of Engineering And Science (IJES),* 3(6), pp. 15-22.

Weijun, X., Zhiming, W., Wei, Z. & Genke, Y., 2004. Applying Particle Swarm Optimization To Job-Shop Scheduling Problem. *Chinese Journal Of Mechanical Engineering,* 17(3), pp. 437- 441.

Yamada, T. & Nakano, R., 1992. A Genetic Algorithm Applicable to Large-Scale Job-Shop Problem. *Parallel Problem Solving from Nature,* Volume 2, pp. 281-290.

Yamada, T. & Nakano, R., 1997. *Genetic algorithms for job-shop scheduling problems.* London, Proceedings of Modern Heuristic for Decision Support, p. 67 – 81.

Zaher, H., ElSherbieny, M., Ragaa, N. & Sayed, H., 2017. A novel Improved Bat Algorithm for Job Shop Scheduling Problem (0975 – 8887). *nternational Journal of Computer Applications,* 5 April, 164(5), pp. 24-30.

Zhang, W. X., Gao, T. Y., Ma, Q. Y., & Xue, D. J., 2011. An Adaptive Genetic Algorithm for the Flexible Job-shop Scheduling Problem. *IEEE* , pp. 405-409.

Zhang, G., Gao, L. & Shi, Y., 2008. *A genetic algorithm and taboo search for solving flexible job shop schedules.* s.l., computational intelligence and design International symposium, pp. 369-372.

Zhu, C., Chen, Y., & Zhang, C. 2009. A Modified Genetic Algorithm to Due Date of Job Shop Scheduling Problem. *2009 International Symposium on Computer Network and Multimedia Technology (CNMT).* IEEE. doi: 10.1109/CNMT.2009.5374722