**OPTIMASI PENJADWALAN PRODUKSI MENGGUNAKAN METODE *HYBRID PARTICLE SWARM OPTIMIZATION* DAN *GENETIC ALGORITHM***

**PROPOSAL TUGAS AKHIR**



**ANAK AGUNG PUTRA ADNYANA**

**1808561098**

**PROGRAM STUDI INFORMATIKA**

**FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM**

**UNIVERSITAS UDAYANA**

**JIMBARAN**

**2021**

# Lembar **PERSETUJUAN** Proposal Tugas Akhir

Judul : Optimasi Penjdwalan Produksi Menggunakan Metode *Hybrid Genetic Algorithm* dan *Particle Swarm Optimization*

Nama : Anak Agung Putra Adnyana

NIM : 1808561098

Tanggal Seminar :

Disetujui Oleh:

Pendamping Proposal

I Made Widiartha, S.Si., M.Kom.

NIP. 198212202008011008

Mengetahui,

Komisi Seminar dan Tugas Akhir

Program Studi Informatika

FMIPA UNUD

Ketua,

I Gusti Ngurah Anom Cahyadi Putra, ST., M.Cs.

NIP. 198403172019031005

# Kata Pengatar

Proposal penelitian dengan judul “Optimasi Penjdwalan Produksi Menggunakan Metode *Hybrid Genetic Algorithm* dan *Particle Swarm Optimization*” ini disusun dalam rangkaian kegiatan pelaksanaan Tugas Akhir di Program Studi Informatika FMIPA UNUD. Proposal ini disusun dengan harapan dapat menjadi pedoman dan arahan dalam melaksanakan penelitian di atas.

Sehubungan dengan telah terselesaikannya proposal ini, maka diucapkan terima kasih dan penghargaan kepada berbagai pihak yang telah membantu pengusul, antara lain:

1. Bapak Dr. I Ketut Gede Suhartana, S.Kom., M.Kom sebagai koordinator program studi Informatika.
2. Bapak I Made Widiartha, S.Si., M.Kom. selaku dosen pembimbing yang telah memberikan arahan dan masukan selama penyusunan proposal tugas akhir.
3. Bapak-bapak dan ibu-ibu dosen di Program Studi Informatika Fakultas MIPA Universitas Udayana, yang telah meluangkan waktu turut memberikan saran dan masukan dalam penyempurnaan proposal ini.
4. Kawan-kawan di Program Studi Informatika yang telah memberikan dukungan moral dalam penyelesaian proposal ini dan juga kedua orang tua yang telah memberikan dukungan berupa moral maupun material.

Disadari pula bahwa sudah tentu proposal ini masih mengandung kelemahan dan kekurangan. Memperhatikan hal ini, maka masukan dan saran-saran penyempurnaan sangat diharapkan.

Jimbaran, April 2020

Penulis

# Daftar Isi

[Lembar PERSETUJUAN Proposal Tugas Akhir i](#_Toc100962243)

[Kata Pengatar ii](#_Toc100962244)

[Daftar Isi iii](#_Toc100962245)

[Daftar Gambar v](#_Toc100962246)

[Daftar Tabel vi](#_Toc100962247)

[BAB I PENDAHULUAN 1](#_Toc100962248)

[1.1 Latar Belakang 1](#_Toc100962249)

[1.2 Rumusan Masalah 2](#_Toc100962250)

[1.3 Tujuan Penelitian 3](#_Toc100962251)

[1.4 Batasan Masalah 3](#_Toc100962252)

[1.5 Manfaat Penelitian 4](#_Toc100962253)

[1.6 Sistematika Penilisan 4](#_Toc100962254)

[BAB II TINJAUAN PUSTAKA 5](#_Toc100962255)

[2.1. Tinjauan Empiris 5](#_Toc100962256)

[2.1.1. HYBRID ALGORITHM AS ALTERNATIVE METHOD FOR OPTIMIZATION, A COMBINATION GENETIC ALGORITHM AND PARTICLE SWARM OPTIMIZATION (Mahardhika, 2021) 5](#_Toc100962257)

[2.1.2. A HYBRID PSO/GA ALGORITHM FOR JOB SHOP SCHEDULING PROBLEM (Tang et al, 2010) 5](#_Toc100962258)

[2.1.3. A NEW HYBRID GENETIC ALGORITHM FOR JOB SHOP SCHEDULING PROBLEM (Wang et al, 2012) 6](#_Toc100962259)

[2.1.4. PERBANDINGAN ALGORITMA GENETIKA DAN PARTICLE SWARM OPTIMIZATION DALAM OPTIMASI PENJADWALAN MATAKULIAH (Marbun dkk, 2013) 6](#_Toc100962260)

[2.1.5. A HYBRID PSO-GA ALGORITHM FOR JOB SHOP SCHEDULING IN MACHINE TOOL PRODUCTION ( Liu et al, 2015) 7](#_Toc100962261)

[2.2. Tinjauan Teori 8](#_Toc100962262)

[2.2.1 Job Shop Scheduling Problem (JSSP) 8](#_Toc100962263)

[2.2.2 Genetic Algorithm 9](#_Toc100962264)

[2.2.3 Particle Swarm Optimization 10](#_Toc100962265)

[BAB III Metodologi Penelitian 13](#_Toc100962266)

[3.1. Data Penelitian. 13](#_Toc100962267)

[3.2. Variabel Penelitian 13](#_Toc100962268)

[3.2.1. Variabel bebas 13](#_Toc100962269)

[3.2.2. Variabel Terikat 14](#_Toc100962270)

[3.3. Tahapan Optimasi Penjadwalan. 14](#_Toc100962271)

[3.4. Metode Perancangan 18](#_Toc100962272)

[3.5. Pengujian 18](#_Toc100962273)

[BAB IV Hasil Dan Pembahasan 20](#_Toc100962274)

[4.1. Gambaran umum sistem 20](#_Toc100962275)

[4.2. Implementasi Sistem 20](#_Toc100962276)

[4.3. Pemrosesan Data 21](#_Toc100962277)

[4.4. Fungsi Objektif 21](#_Toc100962278)

[4.5. Implementasi *Hybrid Genetic Algorithm* dan *Particle Swarm Optimization* 22](#_Toc100962279)

[2.1. Jadwal Pelaksanaan Penelitian 24](#_Toc100962280)

[Daftar Pustaka 25](#_Toc100962281)

# Daftar Gambar

[Gambar 2.1 Contoh Job Shop Scheduling Problem (JSSP) 9](file:///C:\Users\Demniam\Downloads\1808561098_Anak%20Agung%20Putra%20Adnyana_TA.docx#_Toc100962690)

[Gambar 3.1 Flowchart Hybrid PSO-GA 17](#_Toc100962699)

[Gambar 3.2 Alur Penelitian 18](file:///C:\Users\Demniam\Downloads\1808561098_Anak%20Agung%20Putra%20Adnyana_TA.docx#_Toc100962700)

# Daftar Tabel

[Tabel 1.1 Komparasi GA-PSO 2](#_Toc100962933)

[Tabel 4.1 Source Code Fungsi Objektif 21](#_Toc100962943)

[Tabel 4.2 Source Code inisialisasi awal 22](#_Toc100962944)

# BAB I PENDAHULUAN

# Latar Belakang

Semakin berkembangnya persaingan global para pemilik perusahaan akan berlomba-lomba untuk memenangkan persaingan. Dalam dunia industri, proses produksi merupakan proses pembuatan barang atau jasa. Pada proses ini diperlukan strategi dalam menentukan penempatan urutan job sehingga waktu yang diperlukan menjadi efisien. Dengan waktu yang efisien tersebut diharapkan industri dapat mengurangi pengeluaran biaya produksi serta dapat memenuhi kebutuhaan konsumen tepat waktu dalam hal ini industri sering menjumpai hambatan pada aliran proses produksi dikarenakan kurangnya efektifitas dari setiap mesin, dikarenakan, pengalokasian mesin yang bekerja dengan kurang sempurna sehingga terdapat pembagian kerja yang tidak merata pada saat proses produksi. Oleh karena itu pemahaman mengenai konsep penjadwalan sangat penting, sehingga para pekerja mengetahui kapan waktu harus memulai suatu pekerjaan dan kapan waktu mengakhirinya. Penjadwalan disusun dengan mempertimbangkan berbagai batasan yang ada. Penjadwalan yang baik akan memberikan dampak positif, yaitu rendahnya biaya operasi dan waktu pengiriman, yang akhirnya dapat meningkatkan kepuasan pelanggan.

konsep penjadwalan merupakan salah satu contoh dari permasalahan dunia nyata dari *Scheduling Problem* (SP), SP adalah masalah yang berkaitan dengan pengurutan pemrosesan sejumlah pekerjaan pada sejumlah mesin. SP merupakan permasalahan yang memiliki karakteristik terdiri dari m mesin dan n pekerjaan. Setiap pekerjaan harus diproses pada setiap mesin. Masing-masing mesin dapat memroses paling banyak satu pekerjaan pada suatu waktu. Setiap pekerjaan harus diproses pada suatu mesin selarna suatu periode waktu tertentu tanpa interupsi. Setiap pekerjaan hanya dapat diproses oleh satu mesin dalam satu waktu.

Untuk menyelesaikan permasalahan SP dapat digunakan metode *Swarm Intelegence* seperti Metode *Ant Colony Optimization* (ACO), *Artificial Bee Colony (*ABC), *Particle Swarm Optimization* (PSO), dll. Dalam penelitian (Li et al, 2010) 2 buah metode *swarm intelegence* GA dan PSOdianalisa dan dibandingkan, Hasil eksperimen simulasi menunjukkan bahwa GA memiliki kemampuan pencarian global yang kompleks, tetapi efisiensi komputasi rendah dan kecepatan optimasi lambat, sulit untuk konvergen, dan kompleksitas algoritma tinggi. Sedagkan PSO dapat diimplementasikan secara sederhana, dan kecepatan konvergensinya cepat tanpa terlalu banyak parameter, dan memiliki kemampuan pencarian global yang baik, karena informasi partikelnya satu arah, setiap partikel akan mengingat posisi masa lalu, dan konvergensinya sangat cepat, tetapi pada saat yang sama waktu, mudah untuk jatuh ke dalam optima lokal. Dan dalam penelitian lainnya oleh mahardika (2021) di cantumkan juga kelebihan dan juga kekurangan dari GA dan PSO sebagai berikut:

Tabel 1. 1 Komparasi GA-PSO

|  |  |  |
| --- | --- | --- |
| Item pembanding | Algoritma Genetika | *Particle Swarm Optimization* |
| Kemampuan pencarian global | Tinggi | Rendah |
| Implementasi | Sukar | Sederhana |
| Terjebak di lokal optimum | Jarang | Sering |
| Efisiensi komputasi | Rendah | Tinggi |
| Kecepatan optimasi | Rendah | Tinggi |
| Kompleksitas algoritma | Tinggi | Rendah |
| Konvergensi | Sulit | Mudah |

Karena permasalahan tersebut maka pada penelitian yang akan dilakukan, penulis akan mencoba menerapkan metode Hibridasi dari *Particle Swarm Optimization* (PSO) dan *Genetic Algorithm* (GA) dalam menyelesaikan permasalahan *job shop scheduling problem* di penjadwalan produksi industri. Metode ini dipilih karena hasil penelitian Mahardika (2021) menyebutkan bahwa metode *hybrid* ini merupakan alternatif dari metode optimasi dengan hasil yang lebih baik dari GA ataupun PSO.

# Rumusan Masalah

Berdasarkan latar belakang di atas, maka dapat dirumuskan beberapa permasalahan sebagai berikut :

1. Bagaimana mengoptimalkan penjadwalan produksi dengan menggunakan metode *hybrid* PSO GA*.*
2. Bagaimana cara mendapatkan nilai optimal dari parameter dalam PSO dan GA untuk mendapatkan waktu pengerjaan (*makespan*) minimum.
3. Seberapa Efisien jadwal yang di hasilkan menggunakan metode *hybrid* PSO GA.

# Tujuan Penelitian

Berdasarkan rumusan masalah di atas, tujuan penelitian ini adalah sebagai berikut:

1. Mengimplementasikan metode hybrid PSO GA kedalam permasalahan penjadwalan di industri.
2. Mendapatkan solusi jadwal produksi yang optimal.

# Batasan Masalah

Beberapa batasan masalah yang akan dijadikan acuan dalam penelitian ini adalah sebagai berikut :

1. Penelitian ini tidak memperkirakan bahan baku yang tersedia di Gudang.
2. Penelitian ini memperkirakan bahwa semua mesin yang tersedia bekerja, tidak memperkirakan kerusakan mesin.
3. Penelitian ini membatasi variasi dan jumlah pekerjaan yang di jadwalkan.

# Manfaat Penelitian

Beberapa manfaat yang dapat diambil dari penelitian ini adalah sebagai berikut :

1. Dapat meningkatkan produktifitas produksi industri
2. Dapat menjadi solusi untuk pembuatan jadwal kerja di industri yang menerapkan sistem made to order.
3. Mendapatkan waktu penyelesaian yang singkat dalam jadwal produksi yang dihasilkan.

# Sistematika Penilisan

Adapun sistematika penulisan yang digunakan dalam penelitin ini terbagi menjadi beberapa bab yang dapat dilihat pada penjelasan sebagai berikut.

**BAB I PENDAHULUAN**; bab ini memuat latar belakang masalah, rumusan masalah, batasan masalah, tujuan penelitian, manfaat penelitian, serta sistematika penulisan.

**BAB II TINJAUAN PUSTAKA**; pada bab ini terdapat ulasan terori untuk mendukung pendekatan pemecahan masalah (tinjauan teori) serta telaah hasil penelitian-penelitian terdahulu yang berkaitan dengan penelitian yang dilakukan (tinjauan empiris).

**BAB III ANALISIS DAN PERANCANGAN SISTEM**; pada bab ini penulis mengukakakan data yang digunakan serta metode yang digunakan dalam pengumpulan data, desain sistem yang dibuat serta desain evaluasi sistem yang akan digunakan.

**BAB IV HASIL DAN PEMBAHASAN**; pada bab ini, penulis menguraikan implementasi sistem serta hasil penelitian dan analisis yang diperoleh.

**BAB V SIMPULAN DAN SARAN**; bab ini memuat kesimpulan dari hasil penelitian yang telah dilakukan serta saran dari penulis tentang kemungkinan pengembangan dan pemanfaatan hasil penelitian lebih lanjut.

# BAB II TINJAUAN PUSTAKA

Bagian ini menjelaskan mengenai tinjauan empiris dan tinjauan teoritis yang akan dijadikan acuan dalam membangun dan mengimplementasikan Optimasi Sistem Penjdwalan Produksi Menggunakan Metode *Hybrid Genetic Algorithm* dan *Particle Swarm Optimization*.

## Tinjauan Empiris

Pada penelitian ini, terdapat beberapa penelitian sebelumnya yang terkait dengan *Job Shop Scheduling Problem* ataupun metode *Hybrid* GA PSO :

### HYBRID ALGORITHM AS ALTERNATIVE METHOD FOR OPTIMIZATION, A COMBINATION GENETIC ALGORITHM AND PARTICLE SWARM OPTIMIZATION (Mahardhika, 2021)

Penelitian ini membahas tentang perbandingan antara metode hybrid GA PSO dengan metode GA dan metode PSO. Hasil yang didapatkan dari percobaan dengan 3 buah function adalah metode hybrid GA PSO memiliki hasil yang lebih baik dari metode GA maupun metode PSO. Berdasarkan dari penelitian maka penulis menggunakan metode hybrid GA PSO dalam penelitiannya.

### A HYBRID PSO/GA ALGORITHM FOR JOB SHOP SCHEDULING PROBLEM (Tang et al, 2010)

Penelitian ini membahas tentang penyelesaian masalah JSSP menggunakan Algoritma Hybrid PSO dan GA. Percobaan Untuk menggambarkan kinerja PSO / GA untuk JSSP, penulis melakukan pengujian sebanyak 7 benchmark dengan ukuran berbeda. Dari penelitian ini algoritma PSO / GA memiliki stabilitas yang relatif lebih baik. Dan operasi genetik evolusioner yang diusulkan mengarah pada solusi yang lebih baik daripada GA. Hasil eksperimen memverifikasi keefektifan algoritma PSO/GA sehingga mendapatkan solusi terbaik dari yang ditemukan atau solusi Sub-optimal untuk semua benchmark yang dikutip.

### A NEW HYBRID GENETIC ALGORITHM FOR JOB SHOP SCHEDULING PROBLEM (Wang et al, 2012)

Penelitian ini membahas tentang penyelesaian JSSP secara lebih efektif, operator seleksi campuran berdasarkan nilai kebugaran dan konsentrasi yang dirancang untuk meningkatkan keanekaragaman penduduk. Operator crossover berdasarkan mesin, dan operator mutasi berdasarkan critical path (CP) dirancang sesuai dengan model grafik JSSP. Untuk menghitung CP, algoritma baru telah dikirim. Operator Local Search dirancang untuk meningkatkan kualitas solusi. Berdasarkan ini, *Hybrid Genetic Algorithm* diusulkan, dan konvergensi HGA ke solusi global optimum dalam generasi terbatas dengan probabilitas 1 sudah terbukti. Kemudian konvergensi HGA ke solusi optimal dengan probabilitas 1 sudah terbukti. Hasil percobaan menunjukkan bahwa algoritma yang diusulkan efektif dan berkinerja lebih baik daripada algoritma lain yang dibandingkan.

### PERBANDINGAN ALGORITMA GENETIKA DAN PARTICLE SWARM OPTIMIZATION DALAM OPTIMASI PENJADWALAN MATAKULIAH (Marbun dkk, 2013)

Penelitian ini membahas perbandingan antara metode GA dan PSO dalam penyelesaian JSSP. Di penelitian ini metode GA berhasil menyelesaikan penjadwalan matakuliah di prodi Teknik Informatika dengan parameter pengujian yang sudah ditetapkan (parameter GA dan PSO berbeda), Berdasarkan parameter itu, diperoleh nilai fitness 1,0 dengan artian tidak ditemukan bentrok dalam kasus penjadwalan. Hasil ini diperoleh pada iterasi ke 10, dengan waktu eksekusi 8,79 detik. Sedangkan PSO belum berhasil memecahkan permasalahan penjadwalan matakuliah di prodi Teknik Informatika. Berdasarkan parameter pengujian, diperoleh nilai fitness 0,111 dengan dengan 7 bentrokan dalam kasus penjadwalan. Hasil ini diperoleh pada iterasi ke 50, dengan waktu eksekusi 41,636 detik.

### A HYBRID PSO-GA ALGORITHM FOR JOB SHOP SCHEDULING IN MACHINE TOOL PRODUCTION ( Liu et al, 2015)

Penelitian ini membahas tentang implementasi Model pengoptimalan multi-objective berbasis PSO telah disiapkan dalam penelitian kami sebelumnya untuk menyelesaikan JSSP di produksi alat mesin. Solusi PSO melakukan fenomena ketidakseimbangan antara tingkat konvergensi dan konvergensi. Karena operator matematika yang melekat pada algoritma PSO dan pengaturan parameter awal secara praktis aplikasi. Algoritma HPGA diusulkan untuk memecahkan masalah khusus ini, Algoritma yang diusulkan telah mencapai hasil yang bermanfaat dibandingkan dengan tiga metode lain yang umum digunakan. Namun, karena lingkungan penjadwalan produksi terlalu kompleks, studi akan tetap memiliki kesempitan tertentu meskipun dengan penelitian lebih lanjut secara mendalam. Menurut masalah yang tercermin dari tahap analisis dan dari pengalaman.

Meskipun model sistem masih dalam tahap percobaan, namun algoritma yang diusulkan dapat dengan mudah dilakukan tertanam ke dalam model penjadwalan dan mengoptimalkan proses aktual aplikasi perusahaan. Namun karena sifat dari algoritma itu sendiri, maka waktu komputasi akan meningkat secara drastis dengan jumlah yang besar operasi dan karenanya pelanggan tidak dapat puas dalam beberapa aplikasi praktis. Oleh karena itu, penelitian lebih lanjut terhadap algoritma diperlukan dan akan menjadi pekerjaan masa depan.

Dari ketiga penelitian di atas maka penulis akan melakukan penelitian dengan menggabungkan beberapa metode dan algoritma untuk mendapatkan tingkat akurasi deteksi yang baik. Penelitian yang akan dilakukan penulis yaitu menggunakan dataset NSL-KDD yang akan dilakukan seleksi fitur dengan metode information gain lalu akan dilakukan diskritasi terhadap variabel kontinu dengan algoritma K-Means serta akan diklasifikasi menggunakan metode Random Forest yang nantinya akan menghasilkan tingkat akurasi pendeteksian terhadap data normal atau tidak normal.

## Tinjauan Teori

### Job Shop Scheduling Problem (JSSP)

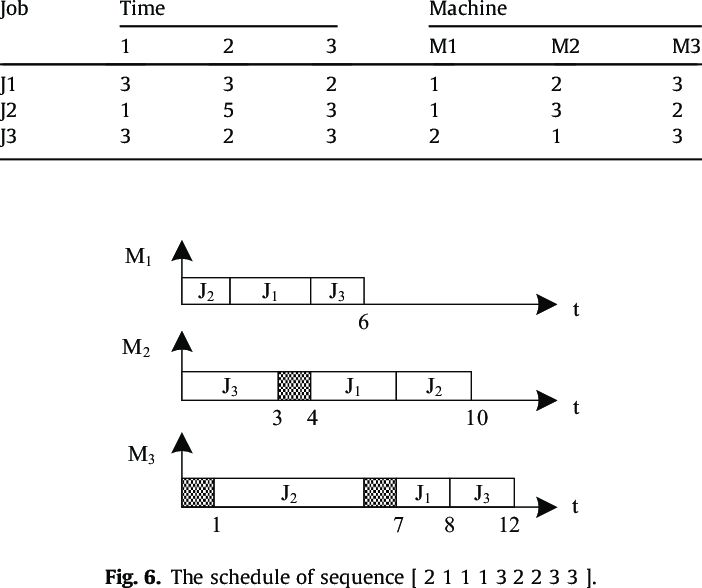
*Job Shop Scheduling Problem* (JSSP) merupakan sebuah masalah penjadwalan yang banyak digunakan pada industri dengan tipikal pesanan adalah made to order, barang pesanan akan dibuat hanya jika terdapat pesanan dari pelanggan. Penjadwalan JSSP merupakan tipe atau jenis penjadwalan yang tergolong rumit, salah satu faktor yang menyebabkan adalah jenis produk atau variasi produk yang ditangani oleh metode ini sangat bervariasi. Banyaknya variasi dari pesanan menyebabkan muncul banyak jenis pekerjaan serta kebutuhan penggunaan alat yang berbeda. Variasi pesanan dan jenis pekerjaan merupakan salah satu kendala dari penyusunan penjadwalan menggunakan metode JSSP. Penyelesaian metode penjadwalan lebih banyak menggunakan pendekatan waktu masuknya pekerjaan “*first come first serve*” atau waktu penyelesaian pekerjaan “*due date*” dan ratio. Dalam beberapa kasus penjadwalan, metode ini akan menimbulkan makespan (waktu pengerjaan) yang tinggi. JSSP dapat dijelaskan sebagai berikut:

Terdapat n pekerjaan yang harus diselesaikan selama rentang waktu [T1, T2]. Pekerjaan ini akan diproses pada m mesin dengan prosedur pemesinan yang diberikan. Setiap pekerjaan dapat diproses pada satu dan hanya satu mesin pada satu waktu dan setiap mesin dapat memproses satu dan hanya satu pekerjaan pada satu waktu. Waktu pemrosesan setiap pekerjaan pada setiap mesin adalah tetap dan diasumsikan telah diketahui sebelumnya. Adapun komponen-komponen pembentuk permasalah JSSP adalah sebagai berikut:

* Rentang waktu pengerjaan [T1, T2], biasanya dalam satuan hari, minggu, atau bulan. (T1 adalah waktu pekerjaan di terima, T2 adalah batas waktu pengerjaan)
* Pekerjaan , n adalah jumlah dari seluruh pekerjaan.
* Mesin , m adalah dari seluruh mesin.
* Matriks urutan pengerjaan untuk setiap pekerjaan , dapat digambarkan sebagai berikut:

Dimana adalah urutan pengerjan dari job ke-I di mesin ke-k

* Matriks waktu pengerjaan untuk setiap pekerjaan , dapat digambarkabn sebagai berikut:

Dimana adalah waktu yang di butuhkan untuk pengerjan job ke-I di mesin ke-k

Gambar 2. 1 Contoh Job Shop Scheduling Problem (JSSP)

### Genetic Algorithm

*Genetic Algorithm* diusulkan oleh Golberg sebagai metode pencarian heuristik yang menyalin seleksi alam. Pada awalnya GA akan memilih secara acak sejumlah solusi individu. Sejumlah individu ini akan dianggap sebagai populasi dan menjadi generasi pertama. Setiap individu memiliki nilai *fitness*. Nilai *fitness* yang semakin tinggi akan membuat individu tersebut menjadi orang tua bagi generasi penerusnya. Sepasang orang tua akan mendapatkan persilangan (*Crossover*) untuk membuat keturunan baru dari mereka. Keturunan ini menjadi anak-anak mereka. Beberapa anak-anak ini mengalami mutasi. Anak-anak ini menjadi individu baru yang akan menjadi orang tua baru bagi generasi selanjutnya. Ini akan berulang sampai beberapa generasi. GA akan berhenti jika solusi ditemukan atau generasi telah mencapai puncaknya (Mahardhika, 2021**)**. Ini adalah proses dari Algoritma Genetika.

* 1. Representasi kromosom.
  2. Pembangunan populasi generasi pertama.
  3. *Benchmarking* setiap individu pada populasi.
  4. Jika nilai *benchmarking* terbaik memecahkan masalah atau generasi telah mencapai nilai maksimalnya lalu proses **BERHENTI**.
  5. Proses seleksi.
  + Hitung nilai *fitness* masing-masing individu.
  + *Elitism*, Pilih 2 individu terbaik yang akan menjadi generasi penerus.
  1. *Crossover*, Pilih 2 individu lalu nikahkan 2 individu tersebut untuk dijadikan 2 anak untuk generasi selanjutnya, proses ini diulang sampai jumlah penduduk terpenuhi.
  2. *Mutation,* Ubah *kromosom* populasi.
  3. Menerapkan individu terbaik dari populasi menjadi generasi penerus, dan ulangi dari proses ke-3.

### Particle Swarm Optimization

*Particle Swarm Optimization* atau yang biasa disingkat PSO merupakan salah satu metode optimisasi heuristic yang biasa digunakan. *Particle Swarm Optimization* terinspirasi oleh pola berkelompok burung atau ikan. PSO ditemukan oleh Russell Eberhart dan James Kennedy di 1995. Katakanlah kita memiliki populasi burung secara acak. Burung yang paling dekat dengan makanan akan berkicau lebih keras dari lainnya, burung ini menjadi posisi kelompok terbaik. Setiap burung memiliki *inertia* kemampuan untuk bergerak. Juga setiap burung memiliki posisi terbaik pribadi untuk makanannya. Jadi kecepatan tiap burung adalah kombinasi informasi dari 3 hal seperti posisi terbaik pribadi, posisi terbaik grup dan inertia kemampuan untuk bergerak. Posisi selanjutnya adalah penjumlahan kecepatan dan posisi sebelumnya. Burung ini akan terus berlanjut berpindah sampai sumber makanan ditemukan atau burung ini mati (Mahardhika, 2021).

Inilah fungsi dari pembaharuan kecepatan:

(1)

Dan untuk fungsi pembaharuan posisi sebagai berikut:

Dengan :

= kecepatan partikel = bilangan random 2 (0-1)

= bobot inertia (*Inertia Weight*) = Pbest (*local optima*)

= *acceleration coefficients* 1 = Gbest (*global optima*)

= *acceleration coefficients* 2 = Partikel saat ini

= bilangan random 1 (0-1)

(2)

Baik R1 dan R2 adalah dua bilangan acak berbeda yang ditarik keluar dari kisaran (0,1). W adalah faktor inersia yang memainkan peran penting untuk menghindari lokal optima. W digunakan untuk menyeimbangkan kinerja pencarian antara eksplorasi global dan lokal dan menghasilkan iterasi yang lebih sedikit untuk menemukan solusi optimal secara efisien.

# BAB III Metodologi Penelitian

Bagian ini akan menjelaskan mengenai langkah-langkah yang akan dilakukan dalam penelitian yang dibuat mengenai Optimasi Penjadwalan produksi menggunakan Metode *hybrid Particle Swarm Optimization* dan *Genetic Algotihm*. Adapun bahasan yang dijelaskan pada bagian ini meliputi data penelitian, Tahap Optimasi Penjadwalan, metode perancangan, dan pengujian dan evaluasi.

## Data Penelitian.

Data yang digunakan dalam penelitian ini berupa data sekunder yang di dapatkan dari penelitian sebelumnya (Purwaningsih dan Fitriana 2016) , data yang akan di gunakan dalam penelitianadalah sebagai berikut:

1. Data mesin.
2. Data order/pekerjaan.
3. Data urutan pengerjaan.
4. Data waktu pengerjaan.
5. Batas waktu pengerjaan

## Variabel Penelitian

Variabel adalah segala sesuatu yang bentuk apa saja yang ditetapkan oleh peneliti untuk dipelajari sehingga diperoleh informasi tentang hal tersebut, kemudian ditarik kesimpulannya.

Dalam penelitian ini, terdapat variable bebas dan variable terikat. Variable-variabel tersebut dapat dijabarkan sebagai berikut:

### Variabel bebas

Variabel bebas adalah variabel yang dapat mempengaruhi hasil dari variabel lain jika nilai dari variabel ini berubah. Dalam penelitian ini, variable bebas adalah dimensi dari data (jumlah data pekerjaan \* jumlah operasi mesin).

### Variabel Terikat

Variabel terikat adalah variabel yang dipengaruhi oleh variabel bebas. Variabel terikat dalam penelitian ini adalah waktu proses.

## Tahapan Optimasi Penjadwalan.

Proses penerapan metode kedalam permasalahan secara tidak langsung di bagi menjadi dua tahap yaitu tahap GA dan juga Tahap PSO, yang nantinya akan di gabungkan menjadi metode hybrid. Berikut merupakan langkah langkahnya:

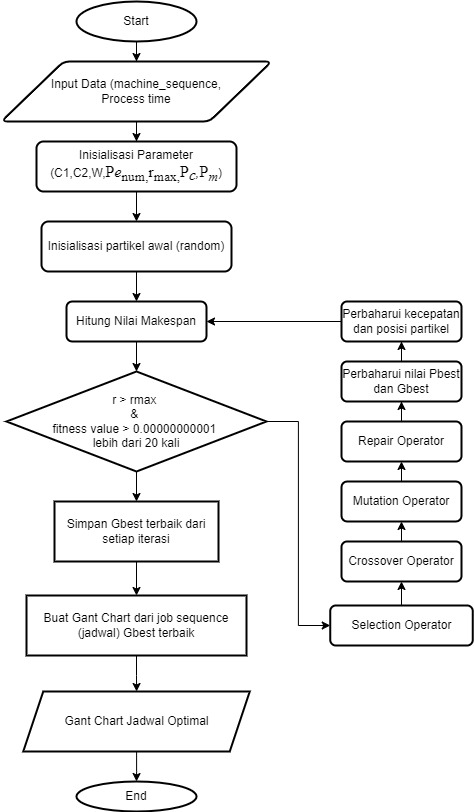
1. Menginputkan data yang akan di proses dalam algoritma, yaitu:

* Seq\_machine adalah matriks urutan pengerjaan.
* pcs\_time adalah matriks waktu pengerjaan.

1. Menginputkan parameter yang akan digunakan yaitu:

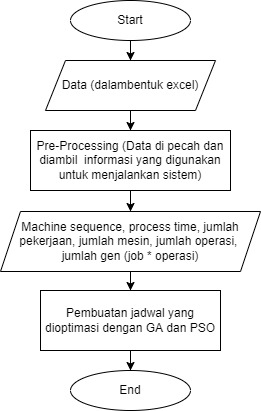
* W adalah faktor inertia.
* C1 adalah koefisien percepatan partikel.
* C2 adalah koefisien percepatan populasi.
* P*e*num adalah ukuran populasi.
* rmax adalah penghitung iterasi maks.
* P*c* adalah probabilitas crossover partikel.
* C adalah cross-bit acak.
* P*m* adalah probabilitas mutasi.

1. Inisialisasi populasi partikel dengan posisi dan kecepatan acak pada dimensi-D di ruang pencarian, setiap partikel mewakili posisi kandidat. Sebuah partikel dianggap sebagai titik dalam ruang dimensi-D, dan statusnya dicirikan menurut posisi dan kecepatannya.
2. Hitung nilai makespan untuk setiap partikel populasi pada iterasi r. (setiap iterasi dilakukan perhitungan nilai makespan)
3. Update nilai optimal dari setiap partikel. P*i* best adalah nilai terbaik untuk partikel i selama proses iterasi, dan Adalah fitness partikel i saat ini. Jika < P*i* best ditetapkan, lalu set P*i* best = , jika tidak, P*i* best tetap.
4. Update nilai optimal dari populasi. Petakan posisi setiap partikel ke dalam ruang solusi dan mengevaluasi nilai kesesuaiannya sesuai dengan fungsi evaluasi pengoptimalan yang diinginkan. Tentukan gbest sebagai nilai terbaik dari file populasi partikel, dan gbest = min (P*i* best), (nilai i = 1, ..., P*e*num).
5. Selama perulangan iterasi jika r <= rmax dan syarat lainya terpenuhi, maka lanjutkan ke Langkah 7, jika tidak, hentikan algoritma dan hitung makepan minimum dari jadwal.
6. Operator Crossover. Dua partikel dalam populasi dipilih dengan probabilitas P*c* sebagai partikel induk untuk operasi crossover. Silangkan induk dengan individu terbaik Xbest (nilai Xbest awal adalah nilai gbest), dan kemudian dua individu terbaik dipilih sebagai solusi. Partikel yang dibutuhkan dalam operasi crossover dibagi menjadi beberapa sub-kelompok. Setiap sub-kelompok mengandung dua partikel dan dilambangkan sebagai X dan Y. Gunakan cross-bit untuk menghasilkan bilangan acak dan . Jika kondisi < Pc2 terpenuhi, maka partikel X akan bertukar gen antara X*i* dan Xbest di posisi yang sama; jika < Pc2 memenuhi, lalu gen antara Y*i*dan Xbest juga ditukar di posisi yang sama.
7. Operator Mutation. jika velocity hampir sama dengan 0, pilih partikel dengan probabilitas Pm dari populasi, dan kemudian hasilkan partikel baru (child) dengan menukar gen dari individu yang dipilih secara acak. Operator ini dapat secara signifikan meningkatkan keragaman populasi dan menghindari masalah lokal optima. Jika kecepatan menjauh dari 0, lanjut ke langkah 11 untuk memperbarui setiap partikel.
8. Operator *Repair*. Operator ini digunakan untuk memeriksa kelayakan individu di populasi. Operator *Repair* digunakan untuk memperbaiki individu yang tidak layak dari populasi menjadi layak. Ide tentang operator ini adalah untuk memisahkan individu yang layak dalam suatu populasi dari mereka yang tidak layak, dan mengembangkan bersama populasi individu yang tidak layak sampai mereka menjadi layak. Cara spesifik untuk melakukan langkah ini adalah menemukan hal yang sama gen dari partikel X dan Y dalam subkelompok dan kemudian menghapusnya. Karenanya, gen baru Z*X* dan Z*Y* dihasilkan. Gen yang hilang harus ditemukan dari individu Xbest terbaik saat ini. Jaga posisi relatif gen tidak berubah, dan kemudian menempelkan gen yang hilang di akhir gen yang baru dihasilkan Z*X* dan Z*Y*.
9. Update kecepatan dan posisi, hal tersebut diperbarui sesuai dengan persamaan (1) dan persamaan (2). Kembali ke Langkah 3 setelah memperbarui kecepatan dan posisi dan memulai iterasi baru. Berikut merupakan flowchart dari gabungan kedua metode:



Gambar 3. 1 Flowchart Hybrid PSO-GA

## Metode Perancangan

Pada bagian ini dijelaskan alur dari penelitian yang akan dilakukan penulis, yaitu mulai dari pengumpulan data, data tersebut akan dilakukan tahap preprocessing, selanjutnya dilakukan proses optimasi penjadwalan yang akan menghasilkan jadwal dengan nilai *fitness* dan *makespan* yang sesuai. Pada tahapan akhir akan dilakukan evaluasi terhadap metode yang digunakan. Adapun alur metodologi penelitian sebagai berikut.

Gambar 3. 2 Alur Penelitian

## Pengujian

Pada penelitian ini akan dilakukan pengujian menentukan solusi optimal dengan beberapa parameter antara lain:

* koefisien percepatan partikel (C1).
* koefisien percepatan populasi (C2).
* probabilitas crossover.
* probabilitas mutation.

Skenario pengujian akan dilakukan dengan 2 cara yaitu:

* + - 1. Menggunakan sejumlah parameter diatas untuk mengetahui nilai parameter yang dapat menghasilkan solusi optimal (makespan terkecil).
      2. Setelah mendapatkan waktu pengerjaan optimal, akan dihitung biaya produksi berdasarkan waktu pengerjaan, biaya penggunaan listrik, biaya penggunaan air, dan gaji pegawai. Menggunakan persamaan sebagai berikut:

(4)

Keterangan:

TC : keseluruhan jumlah biaya produksi yang dikeluarkan oleh perusahaan untuk mendanai aktivitas produksi (Rp)

* + - 1. Lalu setelah di dapatkan solusi optimal maka di lakukan pengujian *collision* (bentrokan jadwal) untuk menilai kinerja dari solusi, dapat digunakan persamaan sebagai berikut:

…………………………………… (5)

Keterangan:

P(N) : Fungsi objektif

iN : Jumlah job yang tidak terjadi *collision* (tabrakan)

N : Jumlah semua job

# BAB IV Hasil Dan Pembahasan

Pada bagian ini dibahas hasil pengembangan sistem optimasi penjadwalan menggunakan hibridasi dua metode *genetic algorithm* dan *Particle Swarm Optimization* dan juga pengujian sistem yang telah dikembangkan.

## Gambaran umum sistem

Penerapan metode hibridasi *Genetic Algorithm* dan *Particle Swarm Optimization* dalam sistem optimasi penjadwalan produksi ini dibangun dengan berbasis desktop. Sistem ini dibangun untuk membantu pengguna yaitu pemilik usaha industri (pabrik) untuk menentukan rekomendasi jadwal yang optimal kepada pekerja/kaywawan-nya agar diharapkan waktu proses pengerjaan dapat ditekan dan menghasilkan biaya pengerjaan (*cost labour*) yang minimal.

Data yang digunakan untuk membangun sistem ini berupa data waktu pengerjaan (*process time*) dari pekerjaan (*job*) yang diterima oleh perusahaan, data urutan operasi pengerjaan mesin (*machine sequence*) dari pekerjaan. Jumlah dari Data digunakan untuk menentukan jumlah dimensi partikel yang akan di hasilkan dan digunakan dalam proses perhitungan algoritma. Besarnya dimensi akan mempengaruhi waktu proses perhitungan, semakin banyak data semakin lama waktu yang dibutuhkan.

Keluaran sistem yang dihasilkan berupa jadwal pengerjaan semua *job* yang ada. Keluaran sistem yang ingin dicapai adalah rekomendasi jadwal produksi yang optimal untuk pekerja. Untuk mendapatkan hasil penjadwalan yang optimal digunakan metode hibridasi *Genetic Algorithm* dan *Particle Swarm Optimization* sebagai pembentuk jadwal produksi dari perusahaan industri.

## Implementasi Sistem

Pada proses pengembangan sistem, sistem ini diimplementasikan menggunakan bahasa pemrograman *Python* dengan versi 3.9 dengan menggunakan *PyCharms* sebagai IDE. Adapun *software* beserta *hardware* yang digunakan yaitu sebagai sebrikut:

Hardware

1. Intel Core i5-6200U 2,3 GHz
2. Memory 4 GB
3. SSD 500 GB

Software

1. Pycharms Community Edition
2. Browser (Microsoft Edge)

## Pemrosesan Data

Data yang digunakan pada penelitian ini yaitu berupa data sekunder, data tersebut didapatkan dari penelitian (purwaningsih dkk) yang merupakan data dari PT.Deco Bali. Didapatkan data berupa waktu pengerjaan beserta urutan pengerjaan pekerjaan, dimana data tersebut sebagai berikut:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| No | Job | Mesin yang digunakan | | |
| operasi 1 | operasi 2 | operasi 3 |
| 1 | Mini bar bv | 1 | 3 | 5 |
| 2 | Wardrobe bv | 2 | 4 | 6 |
| 3 | Luggage bench bv | 1 | 3 | 5 |
| 4 | Console bv | 2 | 4 | 5 |
| 5 | Side table bv | 1 | 3 | 7 |
| 6 | Bed frame bv | 2 | 4 | 5 |
| 7 | Head board bv j-07 | 1 | 3 | 8 |
| 8 | Tv cabinet bv | 2 | 4 | 7 |
| 9 | Daybed bv | 1 | 3 | 8 |
| 10 | Vanity for beach villa | 2 | 4 | 9 |
| 11 | Study table | 1 | 3 | 6 |
| 12 | Study chair | 2 | 4 | 8 |
| 13 | Mirror | 1 | 3 | 10 |
| 14 | Outdoor table | 2 | 4 | 10 |
| 15 | Outdoor chair | 1 | 3 | 11 |
| 16 | Outdoor sofa | 2 | 4 | 9 |
| 17 | Sun bed | 1 | 3 | 11 |
| 18 | Waste bin for minibar | 2 | 4 | 6 |
| 19 | Ladder for towel hanger | 1 | 3 | 9 |
| 20 | Stool for bathroom | 2 | 4 | 6 |
| 21 | Artwork for walkway | 1 | 3 | 6 |
| 22 | Pots for outer walkway | 2 | 4 | 10 |
| 23 | Umbrella holder | 1 | 3 | 10 |
| 24 | Bv wall panel 1 | 2 | 4 | 5 |
| 25 | Bv wall panel 2 | 1 | 3 | 5 |
| 26 | Bench for vanity | 2 | 4 | 9 |
| 27 | Sun umbrella | 1 | 3 | 9 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| No | Job | waktu proses | | |
| operasi 1 | operasi 2 | operasi 3 |
| 1 | Mini bar bv | 9 | 10 | 13 |
| 2 | Wardrobe bv | 13 | 10 | 17 |
| 3 | Luggage bench bv | 6 | 7 | 11 |
| 4 | Console bv | 7 | 6 | 11 |
| 5 | Side table bv | 14 | 15 | 19 |
| 6 | Bed frame bv | 10 | 10 | 12 |
| 7 | Head board bv j-07 | 9 | 9 | 14 |
| 8 | Tv cabinet bv | 13 | 15 | 20 |
| 9 | Daybed bv | 10 | 9 | 13 |
| 10 | Vanity for beach villa | 21 | 23 | 28 |
| 11 | Study table | 16 | 17 | 23 |
| 12 | Study chair | 9 | 10 | 13 |
| 13 | Mirror | 16 | 16 | 24 |
| 14 | Outdoor table | 15 | 15 | 18 |
| 15 | Outdoor chair | 14 | 15 | 19 |
| 16 | Outdoor sofa | 13 | 14 | 21 |
| 17 | Sun bed | 18 | 17 | 21 |
| 18 | Waste bin for minibar | 15 | 13 | 20 |
| 19 | Ladder for towel hanger | 7 | 6 | 11 |
| 20 | Stool for bathroom | 4 | 5 | 7 |
| 21 | Artwork for walkway | 6 | 6 | 12 |
| 22 | Pots for outer walkway | 27 | 28 | 41 |
| 23 | Umbrella holder | 26 | 30 | 40 |
| 24 | Bv wall panel 1 | 20 | 19 | 25 |
| 25 | Bv wall panel 2 | 18 | 21 | 25 |
| 26 | Bench for vanity | 6 | 5 | 13 |
| 27 | Sun umbrella | 12 | 11 | 25 |

## Fungsi Objektif

Dari data tersebut kemudian dilakukan pemecahan dan perubahan bentuk data menjadi list. Dimana data ini akan di di olah menggunakan fungsi objektif untuk mendapatkan *makespan* (waktu pengerjaan) yang paling minimum yang dihasilkan sistem. Adapun rumus dan *source code* dari fungsi objektif adalah sebagai berikut :

(6)

(7)

|  |
| --- |
| Dengan :  = Fungsi Objektif  = Waktu proses sesuai dengan urutan pengerjaan mesin |

Tabel 4. 1 Source Code Fungsi Objektif

|  |
| --- |
| def calculate\_fit(self,total\_chromosome):  chrom\_fitness= []  total\_fitness = 0  j\_keys = [j for j in range(open.num\_job)]  key\_count = {key: 0 for key in j\_keys}  j\_count = {key: 0 for key in j\_keys}  m\_keys = [j + 1 for j in range(open.num\_mc)]  m\_count = {key: 0 for key in m\_keys}  for i in total\_chromosome:  gen\_t = int(open.pt[i][key\_count[i]])  gen\_m = int(open.ms[i][key\_count[i]])  j\_count[i] = j\_count[i] + gen\_t  m\_count[gen\_m] = m\_count[gen\_m] + gen\_t  if m\_count[gen\_m] < j\_count[i]:  m\_count[gen\_m] = j\_count[i]  elif m\_count[gen\_m] > j\_count[i]:  j\_count[i] = m\_count[gen\_m]  key\_count[i] = key\_count[i] + 1  makespan = max(j\_count.values())  chrom\_fitness.append(1 / makespan)  return sum(chrom\_fitness) |

## Implementasi *Hybrid Genetic Algorithm* dan *Particle Swarm Optimization*

Langkah selanjutnya yang dilakukan dalam mengimplementasikan metode *hybrid Genetic Algorithm* dan *Particle Swarm Optimization* yaitu membentuk kombinasi pekerjaan yang akan dijadikan jadwal pengerjaan dan di hitung nilai dari fungsi objektifnya, Langkah pemrosesan dari metode *hybrid* adalah sebagai berikut :

Langkah pertama dari metode adalah inisialisasi partikel awal. Proses ini mengacak *random* semua partikel (*job*) yang ada dalam data dan menjadikanya pratikel awal.

Gambar 4. 1 Inisialisasi Partikel

Jumlah dari partikel adalah sebanyak 81 partikel (jumlah *job* \* jumlah operasi). Selain partikel, inisialisai kecepatan (*velocity*) awal juga dilakukan dengan rentang nilai 1 – 10, jumlah dari *velocity* sesuai dengan jumlah partikel. *Source code* dari proses inisialisai awal dapat dilihat di tabel 4.2.

Tabel 4.2 Source Code inisialisasi awal

|  |
| --- |
| def \_\_init\_\_(self):  def starting\_pos():  start = np.array([0,0,0,1,1,1,2,2,2,3,3,3,4,4,4,5,5,5,6,6,6,7,7,7,8,8,8,9,9,9,10,10,10,11,11,11,12,12,12,13,13,13,14,14,14,15,15,15,16,16,16,17,17,17,18,18,18,19,19,19,20,20,20,21,21,21,22,22,22,23,23,23,24,24,24,25,25,25,26,26,26])  shuffle(start)  par = list(start)  return par  def starting\_vel():  vel = []  for i in range(open.num\_gene):  vel= np.random.randint(1,10, size=open.num\_gene) return vel |

|  |
| --- |
| def instantantiate(pop):  swarm = []  for i in range(0, pop):  swarm.append(Particle())  return swarm |

Proses ini diulang sebanyak jumlah populasi partikel. Dengain ini didapatkan *swarm* partikel awal, *source code* dan contoh hasil dari pembentukan *swarm* terdapat dalam table 4.3 dan gambar 4.2.

[0, 10, 23, 14, 26, 15, 13, 0, 21, 4, 5, 9, 13, 12, 13, 16, 11, 5, 8, 22, 8, 2, 5, 19, 14, 19, 22, 21, 2, 7, 10, 12, 16, 12, 11, 18, 10, 16, 17, 22, 3, 18, 4, 11, 7, 3, 23, 1, 2, 6, 9, 7, 4, 25, 21, 1, 25, 24, 15, 23, 19, 3, 0, 20, 24, 8, 1, 25, 18, 15, 6, 9, 17, 17, 6, 24, 6, 14, 20, 26, 20]

[25, 16, 9, 12, 0, 21, 0, 16, 8, 7, 18, 15, 19, 15, 22, 9, 18, 15, 13, 23, 12, 24, 8, 13, 6, 1, 24, 3, 3, 0, 5, 4, 23, 22, 18, 20, 7, 24, 11, 6, 16, 9, 12, 23, 20, 25, 10, 25, 20, 7, 26, 19, 6, 26, 21, 17, 14, 10, 11, 14, 10, 4, 17, 13, 17, 14, 2, 19, 4, 1, 3, 21, 2, 11, 5, 1, 26, 2, 5, 22, 8]

[3, 19, 9, 25, 3, 1, 17, 7, 20, 5, 15, 23, 13, 25, 23, 15, 26, 11, 16, 1, 4, 22, 20, 14, 18, 24, 5, 19, 16, 8, 4, 5, 26, 25, 8, 24, 3, 6, 14, 2, 12, 20, 15, 22, 23, 9, 14, 6, 24, 11, 12, 0, 12, 4, 9, 1, 18, 21, 6, 0, 11, 13, 17, 17, 8, 13, 21, 10, 16, 0, 21, 10, 22, 10, 26, 2, 7, 7, 19, 2, 18]

[21, 24, 10, 11, 5, 11, 12, 3, 6, 10, 17, 17, 4, 25, 20, 20, 18, 19, 15, 6, 22, 18, 1, 12, 14, 7, 18, 15, 22, 21, 0, 21, 7, 12, 2, 24, 19, 16, 5, 6, 25, 24, 15, 23, 3, 2, 16, 19, 2, 20, 23, 0, 0, 11, 1, 4, 25, 10, 3, 23, 17, 14, 26, 22, 8, 4, 26, 26, 8, 16, 9, 13, 8, 5, 13, 9, 9, 14, 13, 1, 7]

[12, 26, 5, 9, 8, 22, 18, 26, 6, 0, 3, 23, 17, 17, 0, 19, 13, 11, 8, 21, 12, 1, 22, 1, 17, 2, 22, 1, 11, 10, 24, 3, 9, 6, 23, 16, 13, 2, 21, 15, 21, 9, 0, 15, 14, 18, 6, 13, 11, 16, 3, 25, 20, 23, 12, 2, 26, 19, 5, 10, 25, 4, 4, 24, 14, 8, 20, 19, 15, 25, 4, 18, 5, 16, 10, 14, 7, 7, 7, 24, 20]

[10, 8, 4, 1, 2, 23, 17, 3, 14, 21, 25, 7, 5, 20, 1, 10, 14, 15, 22, 15, 19, 7, 10, 19, 25, 0, 6, 13, 3, 5, 14, 24, 13, 21, 20, 13, 19, 15, 25, 16, 12, 23, 9, 22, 12, 24, 16, 24, 18, 0, 26, 17, 16, 2, 22, 18, 20, 26, 11, 6, 26, 9, 5, 0, 8, 4, 8, 18, 7, 4, 11, 9, 17, 12, 1, 3, 11, 2, 21, 6, 23]

Gambar 4. 2 Contoh hasil dari inisialisasi swarm

[19, 2, 4, 26, 9, 17, 22, 4, 26, 2, 18, 14, 4, 11, 7, 15, 21, 12, 3, 3, 14, 22, 5, 14, 12, 24, 11, 19, 15, 8, 18, 10, 5, 5, 1, 18, 21, 7, 10, 0, 12, 22, 15, 7, 26, 8, 24, 6, 19, 13, 3, 0, 0, 16, 11, 1, 25, 21, 6, 10, 2, 13, 17, 17, 16, 13, 6, 20, 20, 23, 16, 9, 8, 24, 23, 25, 9, 1, 25, 23, 20]

[4, 12, 12, 18, 17, 17, 10, 0, 9, 14, 0, 22, 18, 7, 14, 23, 4, 8, 3, 25, 9, 7, 15, 20, 8, 5, 14, 4, 18, 6, 21, 15, 24, 24, 10, 11, 26, 2, 1, 22, 2, 16, 11, 12, 23, 20, 19, 20, 17, 1, 16, 25, 19, 7, 26, 25, 21, 13, 3, 13, 22, 23, 16, 24, 8, 3, 11, 5, 0, 10, 2, 21, 1, 6, 5, 19, 6, 26, 15, 9, 13]

**……**

[4, 25, 1, 10, 21, 23, 3, 8, 25, 19, 11, 15, 23, 12, 22, 20, 12, 24, 11, 22, 18, 18, 7, 21, 26, 11, 19, 14, 6, 6, 14, 1, 19, 22, 26, 5, 23, 9, 14, 4, 5, 0, 8, 25, 12, 7, 17, 9, 8, 4, 20, 9, 6, 10, 0, 24, 2, 7, 0, 13, 26, 20, 2, 13, 21, 15, 17, 5, 18, 3, 10, 2, 13, 1, 17, 16, 16, 16, 3, 15, 24]

[13, 26, 11, 3, 24, 17, 13, 19, 11, 24, 18, 9, 17, 0, 16, 24, 20, 19, 23, 23, 26, 18, 21, 8, 2, 4, 9, 12, 15, 16, 12, 1, 9, 5, 20, 22, 0, 14, 8, 4, 25, 0, 3, 15, 5, 19, 18, 20, 6, 12, 6, 6, 26, 15, 8, 16, 14, 14, 21, 22, 1, 2, 25, 7, 21, 5, 22, 10, 11, 17, 23, 7, 1, 10, 10, 2, 7, 13, 4, 3, 25]

Setelah melakukan proses inisialisasi awal, maka akan dihitung nilai fitness dari setiap partikel dalam populasi menggunakan fungsi objektif pada tabel 4.1 diatas. Didapatkan hasil seperti pada gambar 4.3.

**Hasil Fungsi Objektif**

1. 0.0030211480362537764
2. 0.00303951367781155
3. 0.002364066193853428
4. 0.0029411764705882353
5. 0.002881844380403458
6. 0.002881844380403458
7. 0.0036900369003690036
8. 0.003048780487804878
9. 0.0027397260273972603
10. 0.0034482758620689655
11. 0.002967359050445104
12. 0.002932551319648094
13. 0.003246753246753247
14. 0.002840909090909091
15. 0.0030303030303030303
16. 0.0031446540880503146
17. 0.003048780487804878
18. 0.0028089887640449437
19. 0.0029498525073746312

Langkah selanjutnya adalah proses *selection* ini merupakan bagian dari *genetic algorithm* proses ini berfungsi untuk memilih 2 buah partikel terbaik sebagai *parents* (induk) untuk digunakan dalam proses *crossover*. metode *selection* yang digunakan dalam sistem ini adalah metode *roulete wheel,* partikel yang memiliki *fitness value* terkecil memiliki kemungkinan lebih tinggi untuk dipilih untuk generasi yang berikutnya. Untuk menghitung probabilitas *fitness* perlu dihitung *fitness* masing-masing partikel. Contoh perhitungan probabilitas fitness dapat dilihat dalam gambar 4.4.

Gambar 4. 3 Contoh Hasil Perihutngan Fungsi Objektif

1. 0.003194888178913738
2. 0.002403846153846154
3. 0.0030581039755351682
4. 0.002421307506053269
5. 0.0028089887640449437
6. 0.002849002849002849

…..

1. 0.003048780487804878

Fitness[0] = 0.0030211480362537764

Fitness[1] = 0.00303951367781155

Fitness[2] = 0.002364066193853428

Fitness[n] = ….

Total = (0.0030211480362537764 + 0.00303951367781155 + 0. 02364066193853428 + …. + n)

Total = 0.0845

P[0] = 0.0030211480362537764 / 0.0845

= 0.0357531360946746

P[1] = 0.00303951367781155 / 0.0845

= 0.0359705325443787

P[2] = 0.002364066193853428/ 0.0845

= 0.0279771147100592

P[n] = … / 0.0845

= …

Gambar 4. 4 Contoh perhitungan probabilitas fitness

Lalu hitung *Cumulative Probability* dari partikel

C[0] = 0.0357531360946746

C[1] = 0.0357531360946746 + 0.0359705325443787

= 0.0717236686390533

C[2] = 0.0357531360946746 + 0.0359705325443787 + 0.0279771147100592

= 0.0997007833491125

C[n] = 0.0357531360946746 + 0.0359705325443787 + 0.0279771147100592 + … + n

= 1

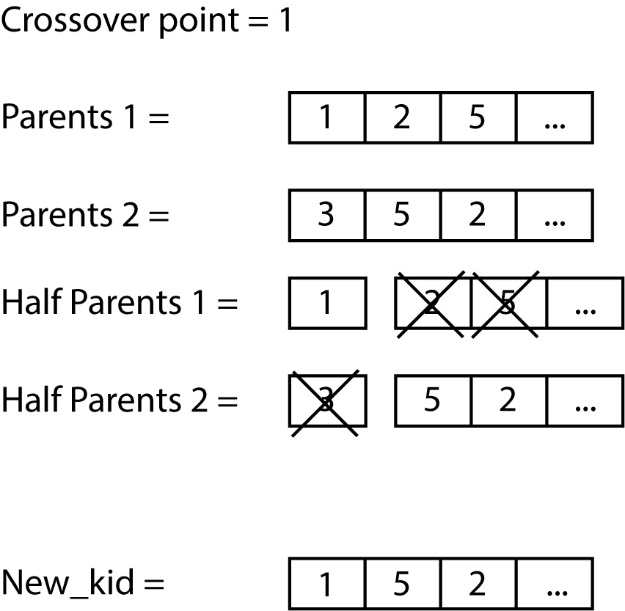
Jika C[i] < R[i] dimana R adalah bilangan random dari 0-1

Maka partikel[i] menjadi *parents*

Pada table 4. Dapat dilihat *source code* dari proses selection.

|  |
| --- |
| def selection(swarm):  fit\_r = []  for particle in swarm:  #print(particle.fit)  fit\_r.append(particle.fit)  # Probability over total probability  fit\_r\_sum = sum(fit\_r)  selection\_probability = []  for relative\_fit in fit\_r:  selection\_probability.append(relative\_fit / fit\_r\_sum)  # Cumulative probability  cumulative\_probability = []  the\_sum = 0  for a in selection\_probability:  the\_sum += a  cumulative\_probability.append(the\_sum)  # For the new generation, we compare a random number between 0 and 1  # and we select the particle that has the next greater cumulative probability  probability = random.random()  for i in range(0, len(cumulative\_probability)):  if probability <= cumulative\_probability[i]:  new\_kid = swarm[i]  break  # Make new copy  a\_new\_kid = Particle()  a\_new\_kid.v = new\_kid.v[:]  a\_new\_kid.x = new\_kid.x[:]  a\_new\_kid.p\_best = new\_kid.p\_best[:]  #print(a\_new\_kid.x)  return a\_new\_kid |

Setelah didapatkan 2 partikel *parents* maka langkah selanjutnya adalah proses *crossover* metode yang digunakan adalah *two point crossover*. Dimana proses ini dilakukan dengan membangkitkan bilangan acak 0 - 1 sejumlah populasi, kemudian bandingkan bilangan acak tersebut dengan Cr (CrossoverRate) yang telah ditentukan. Jika syarat proses *crossover* terpenuhi barulah di jalankan proses *selection*, ini dilakukan untuk mengurangi beban sistem, jikan proses *selection* di lakukan terpisah dengan *crossover*, terjadi proses *redundant* yang mengakibatkan beratnya kinerja *memory*. Kemudian setelah *parents* didapatkan untuk dilakukan *crossover*, maka dibangkitkan bilangan acak dengan rentang 1 – jumlah partikel. Bilangan ini akan digunakan menjadi titik potong dari partikel *parents*. Lalu setelah partikel dipotong, sambungkan patikel *half\_parents* 1 dan juga *half\_parents* 2. Contoh perhitungan dari crossover dapat dilihat pada gambar 4.5, dan *source code* dapat dilaht pada table 4.



Gambar 4. 5 Contoh Crossover

|  |
| --- |
| def crossover(self, other\_particle):  crossover\_position = randint(1, len(self.x))  new1\_first\_half = self.x[:crossover\_position]  new1\_second\_half = other\_particle.x[crossover\_position:]  temp = new1\_first\_half + new1\_second\_half  new = Particle()  new.x = temp  new.v = self.v  new.fit = self.fit  new.p\_best = self.p\_best  return new  # pemanggialn function crossover dan selection pada function utama  for j in range(len(swarm)):  # Decide for crossover  dont\_crossover = random.random()  if dont\_crossover < CROSSOVER\_PROB:  parent1 = selection(swarm)  parent2 = selection(swarm)  a\_new\_kid = parent1.crossover(parent2)  else:  a\_new\_kid = selection(swarm)  a\_new\_kid.mutate()  a\_new\_kid.remove\_duplicates()  new\_gen.append(a\_new\_kid) |

Pada tabel 4. Pada bagian pemanggilan *crossover* jika partikel tidak memenuhi syarat untuk proses *crossover* (*random number* < Cr), maka partikel akan di lanjutkan pada proses berikutnya yaitu proses *mutation*. dimana proses diawali dengan membangkitkan bilangan acak 0 - 1 sejumlah partikel dengan melakukan perulangan sejumlah populasi. Selanjutnya bandingkan bilangan acak tersebut dengan Mr (*Mutation Rate*) yang telah ditentukan. Jika Mr > bilangan acak, maka lakukan proses *mutation* pada partikel [i]. nilai dari partikel [i] akan diganti dengan bilangan acak dengan rentang 1- jumlah *job*. *Source code* dari proses *mutation* dapat dilihat pada tabel 4.

|  |
| --- |
| def mutate(self):  for m in range(0,len(self.x)):  dont\_mutate = random.random()  if Particle.MUTATION\_PROB > dont\_mutate:  self.x[m] = randint(0,26) |

Langkah selanjutnya adalah proses *repair*, setelah melalui proses *selection, crossover,* dan *mutation* terdapat kemungkinan partikel menjadi kacau (*job* muncul dalam *sequence* lebih daripada jumlah operasi). Karena itulah diperlukan proses repair, proses ini akan memperbaiki job sequence dalam partikel agar sesuai dengan syarat dari permasalahan (kemunculan *job* dalam *sequence* sama dengan jumlah operasi). Proses ini dimulai dengan menghitung kemunculan dari *job* dalam *sequence* untuk setiap partikel dalam populasi, kemudian posisi dan jumlah *job* dalam *sequence* disimpan. Lalu dilakukan pengecekan jumlah *job*, jika lebih dari jumlah operasi maka *job* akan dimasukan kedalam list *larger* dan sebaliknya jika *job* kurang dari jumlah operasi maka akan dimasukan kedalam list *less*. Lalu akan dilakukan penukaran job yang terdapat dalam list *larger* dan list *less*, proses ini dilakukan selama jumlah *job* dalam *sequence* belum memenuhi syarat (bernilai sama dengan jumlah operasi). Source code dari proses repair dapat dilihat pada tabel 4.

|  |
| --- |
| def remove\_duplicates(self):  job = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26]  job\_count = []  job\_pos = []  larger = []  less = []  for i in range(0, open.num\_job):  job\_count.append(self.x.count(i))  for j in job:  job\_pos.append(list\_duplicates\_of(self.x, j))  for i in range(0, open.num\_job):  if job\_count[i] >= 3:  larger.append(i)  else:  less.append(i)  for i in range(0, open.num\_job):  x = 0  while job\_count[i] > 3:  if not less:  break  else:  index = random.choice(less)  self.x[job\_pos[i][x]] = index  job\_count[index] = job\_count[index] + 1  job\_count[i] = job\_count[i] - 1  x += 1  if job\_count[index] == 3:  less.remove(index) |

Lalu langkah selanjutnya adalah melakukan pembaharuan *fitness value* dari semua partikel baru yang dihasilkan dari ketiga proses diatas. Langkah dan juga *source code* masih sama seperti langkah diatas. Lalu setelah mendapatkan *fitness value* baru akan dilakukan pencarian Pbest dan Gbest. Proses dimulai dengan mencari Pbest terlebih dahulu, ini dilakukan dengan membandingkan semua *fitness* *value* dari partikel yang ada dalam satu generasi, kemudian nilai Pbest akan disimpan untuk setiap generasi. Dan untuk Gbest nilai dari semua Pbest akan dibandingkan untuk mendapatkannya. Untuk source code dari Pencarian Pbest dan Gbest dapatdilihat dalam tabel 4.

|  |
| --- |
| # Find p\_best of each particle  for particle in swarm:  if particle.fit < particle.calculate\_fit(particle.p\_best):  particle.p\_best = particle.x[:]  # Find g\_best  fitt = []  position = 0  for particle in swarm:  fitt.append(particle.calculate\_fit  (particle.p\_best))  algorithm.minimum = max(fitt)  if algorithm.minimum < particle.calculate\_fit(Particle.g\_best) :  position = fitt.index(algorithm.minimum)  Particle.g\_best = swarm[position].x[:]  algorithm.best\_x = swarm[position].x[:]  best\_history.append(algorithm.minimum)  algorithm.gen = i |

Langkah selanjutnya setelah mencari Pbest dan Gbest adalah proses *update* *velocity* dan *update* *position*, proses ini dilakukan untuk semua partikel selain Gbest, pertama dilakukan proses *update* *velocity*, nilai partikel *velocity* akan di rubah berdasarkan fungsi (1) yang telah dibahas sebelumnya pada bab dua, kemudian dilakukan proses *update* *position* menggunakan fungsi (2) yang telah dibahas juga sebelumnya pada bab dua. Setelah nilai posisi baru didapatkan, nilai tersebut akan di urutkan dari nilai terkecil, kemudian *job* yang merepresentasikan nilai tersebut akan disusun sesuai nilai yang telah diurutkan. Source code dari proses *update* *velocity* dan *update* *position* dapat dilihat pada tabel 4.

|  |
| --- |
| def update\_velocity(self):  """Updates the velocity of each dimension in the particle"""  for i in range(len(self.v)-1):  vel = Particle.W \* self.v[i] + Particle.C1 \* random.random() \* (self.p\_best[i] - self.x[i]) + Particle.C2 \* random.random() \* (Particle.g\_best[i] - self.x[i])  self.v[i] = vel  def update\_position(self):  """Updates the position of each dimension in the particle"""  val = []  for i in range(0, len(self.x)):  temp = [0,0]  new\_pos = float(self.x[i] + self.v[i])  temp= [new\_pos,self.x[i]]  val.append(temp)  val.sort()  for j in range(len(val)):  new\_par = val[j][1]  return new\_par |

## Implementasi *Gantt Chart*

Setelah mendapatkan *job sequence* paling optimal maka jadwal akan dibuatkan *gantt chart* untuk *sequence*, pembuatan *gantt chart* menggunakan bantuan *library chart-studio* yang terdapat dalam python, pada tabel 4. Dapat dilihat source code dari pembuatan *gantt chart*.

|  |
| --- |
| def gant\_chart(G\_sequence\_best):  '''--------plot gantt chart-------'''  j\_keys = [j for j in range(open.num\_job)]  key\_count = {key: 0 for key in j\_keys}  j\_count = {key: 0 for key in j\_keys}  m\_keys = [j + 1 for j in range(open.num\_mc)]  m\_count = {key: 0 for key in m\_keys}  j\_record = {}  today = datetime.datetime.today()  for i in G\_sequence\_best:  gen\_t = int(open.pt[i][key\_count[i]])  gen\_m = int(open.ms[i][key\_count[i]])  j\_count[i] = j\_count[i] + gen\_t  m\_count[gen\_m] = m\_count[gen\_m] + gen\_t  if m\_count[gen\_m] < j\_count[i]:  m\_count[gen\_m] = j\_count[i]  elif m\_count[gen\_m] > j\_count[i]:  j\_count[i] = m\_count[gen\_m]  start\_time = str(today + datetime.timedelta(seconds=(j\_count[i] - open.pt[i][key\_count[i]]) \* 3600)) # convert seconds to hours, minutes and seconds  end\_time = str(today + datetime.timedelta(seconds=j\_count[i] \* 3600))  j\_record[(i, gen\_m)] = [start\_time, end\_time]  key\_count[i] = key\_count[i] + 1  df = []  r = lambda: random.randint(0, 255)  colors = ['#%02X%02X%02X' % (r(), r(), r())]  for j in j\_keys:  colors.append('#%02X%02X%02X' % (r(), r(), r()))  for i in range(3):  m = int(open.ms[j][i])  df.append(dict(Task='Machine %s' %(m), Start='%s' %(str(j\_record[(j, m)][0])), Finish='%s' % (str(j\_record[(j, m)][1])), Resource='Job %s' % (j + 1)))  fig = ff.create\_gantt(df,colors=colors, index\_col='Resource', show\_colorbar=True, group\_tasks=True, showgrid\_x=True,title='Job shop Schedule')  return fig.show() |

## Hasil Pengujian

Berdasarkan implementasi algoritma *Hybrid Genetic Algorithm* dan *Particle Swarm Optimization* yang telah dijelaskan sebelumnya. Selanjutnya dilakukan pengujian untuk melihat apakah implementasi dari algoritma tersebut berjalan atau tidak. Didalam pengujian ini terdapat beberapa parameter yang ditentukan sebagai berikut:

|  |  |
| --- | --- |
| **Parameter** | **Values** |
| Population | 100 |
| Generation | 200 |
| Particle size | 81 (Number of Job\*Number of Operation) |

# Jadwal Pelaksanaan Penelitian

Dalam pelaksanaan penelitian yang dilakukan, penulis memerluka waktu selama lima bulan~~.~~ Dengan rincian kegiatan yang dilakukan dapat dilihat pada table berikut.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| No | Kegiatan | Bulan | | | | |
| 1 | 2 | 3 | 4 | 5 |
| 1 | Studi Literatur |  |  |  |  |  |
| 2 | Pengumpulan Data |  |  |  |  |  |
| 3 | Pengolahan Data |  |  |  |  |  |
| 4 | Perancangan Sistem |  |  |  |  |  |
| 5 | Penerapan Metode |  |  |  |  |  |
| 6 | Pengujian, Analisis dan Evaluasi |  |  |  |  |  |
| 7 | Penulisan Laporan Akhir |  |  |  |  |  |

Tabel 8. 1 Jadwal Pelaksanaan Penelitian

# Daftar Pustaka

Purwaningsih, Ratna, dan Ines Chandra Fitriana. 2016. Analisis Penjadwalan Produk PT Eksotika Logam Bali ( DECO BALI ) dengan Minimasi Makespan, 124–31.

Li, Z., Liu, X., Duan, X., & Huang, F. (2010). Comparative research on particle swarm optimization and genetic algorithm. Computer and information science, 3(1), 120-127.

Liu, L. L., Hu, R. S., Hu, X. P., Zhao, G. P., & Wang, S. (2015). A hybrid PSO-GA algorithm for job shop scheduling in machine tool production. International Journal of Production Research, 53(19), 5755-5781.

Mahardhika, T. (2021, February). Hybrid Algorithm as alternative method for optimization, a combination Genetic Algorithm and Particle Swarm Optimization. In Journal of Physics: Conference Series (Vol. 1764, No. 1, p. 012040). IOP Publishing.

Marbun, Y., Nikentari, N., & Bettiza, M. (2013). Perbandingan Algoritma Genetika dan Particle Swarm Optimization dalam Optimasi Penjadwalan Matakuliah. Fak. Tek. Umr, 1-7.

Syarif, A., Ruby, T., & Saputra, A. (2009, November). Analisis Kinerja Genetic Algorithm Pada Job Shop Scheduling. In Seminar Nasional Sains, Matematika, Informatika dan Aplikasinya VI UNILA (Vol. 1, No. 1, pp. 195-202). Fak. Mipa Universitas Lampung.

Tang, J., Zhang, G., Lin, B., & Zhang, B. (2010, June). A hybrid PSO/GA algorithm for job shop scheduling problem. In International Conference in Swarm Intelligence (pp. 566-573). Springer, Berlin, Heidelberg.

Wang, Y. (2012). A new hybrid genetic algorithm for job shop scheduling problem. Computers & Operations Research, 39(10), 2291-2299.

Wong, T. C., & Ngan, S. C. (2013). A comparison of hybrid genetic algorithm and hybrid particle swarm optimization to minimize makespan for assembly job shop. Applied Soft Computing, 13(3), 1391-1399.