SKRIPSI

ALGORITMA ANT COLONY UNTUK PERMASALAHAN MULTI OBJECTIVE FLOWSHOP SCHEDULING

Kevin Jonathan

NPM: 2014730020

PROGRAM STUDI TEKNIK INFORMATIKA
FAKULTAS TEKNOLOGI INFORMASI DAN SAINS
UNIVERSITAS KATOLIK PARAHYANGAN
2019

UNDERGRADUATE THESIS

ANT COLONY ALGORITHM FOR MULTI OBJECTIVE FLOWSHOP SCHEDULING PROBLEMS

Kevin Jonathan

NPM: 2014730020

DEPARTMENT OF INFORMATICS
FACULTY OF INFORMATION TECHNOLOGY AND SCIENCES
PARAHYANGAN CATHOLIC UNIVERSITY
2019

LEMBAR PENGESAHAN

ALGORITMA ANT COLONY UNTUK PERMASALAHAN MULTI OBJECTIVE FLOWSHOP SCHEDULING

Kevin Jonathan

NPM: 2014730020

Bandung, 10 Desember 2019

Menyetujui,

Pembimbing

Dr.rer.nat. Cecilia Esti Nugraheni

Ketua Tim Penguji	Anggota Tim Penguji
Luciana Abednego, M.T.	**Rosa De Lima, M.Kom.**

Mengetahui,

Ketua Program Studi

Mariskha Tri Adithia, P.D.Eng

PERNYATAAN

Dengan ini saya yang bertandatangan di bawah ini menyatakan bahwa skripsi dengan judul:

ALGORITMA ANT COLONY UNTUK PERMASALAHAN MULTI OBJECTIVE FLOWSHOP SCHEDULING

adalah benar-benar karya saya sendiri, dan saya tidak melakukan penjiplakan atau pengutipan dengan cara-cara yang tidak sesuai dengan etika keilmuan yang berlaku dalam masyarakat keilmuan.

Atas pernyataan ini, saya siap menanggung segala risiko dan sanksi yang dijatuhkan kepada saya, apabila di kemudian hari ditemukan adanya pelanggaran terhadap etika keilmuan dalam karya saya, atau jika ada tuntutan formal atau non-formal dari pihak lain berkaitan dengan keaslian karya saya ini.

Dinyatakan di Bandung,
Tanggal 10 Desember 2019

Meterai
Rp. 6000

Kevin Jonathan
NPM: 2014730020

ABSTRAK

Lorem ipsum dolor sit amet, ex quo duis discere facilis, gloriatur democritum mea ex. Eu periculis voluptatum assueverit eam, in mea oblique adipisci hendrerit, illud aliquando ne ius. Cu invidunt constituam vix, vidisse moderatius reprehendunt his cu. Movet graece ne sit.

Cu omnis homero omnium nec, eu ubique mollis vivendo eos. Et soleat mandamus scriptorem duo. Ne eos alii euripidis sententiae. Soleat quaestio sit id, eam delenit fuisset assueverit an. In mei laboramus voluptatum. Usu cu possim impetus, pri inermis consulatu et, an affert abhorreant vis.

Usu an mediocrem dissentiunt delicatissimi. Cum dolores propriae cu. Ad sonet officiis perpetua vel, at legimus luptatum eos, pro dictas invidunt suscipiantur te. Ut movet dicit eleifend eam, vis ei graece splendide, est nihil splendide ex. Ad minim abhorreant cum, usu ut voluptua intellegat. Et his feugait epicurei, mazim nihil cu vix, nec amet dicam volumus ut.

Kata-kata kunci: Flow Shop, Multi Objective, Penjadwalan, Ant Colony

ABSTRACT

Lorem ipsum dolor sit amet, ex quo duis discere facilis, gloriatur democritum mea ex. Eu periculis voluptatum assueverit eam, in mea oblique adipisci hendrerit, illud aliquando ne ius. Cu invidunt constituam vix, vidisse moderatius reprehendunt his cu. Movet graece ne sit.

Cu omnis homero omnium nec, eu ubique mollis vivendo eos. Et soleat mandamus scriptorem duo. Ne eos alii euripidis sententiae. Soleat quaestio sit id, eam delenit fuisset assueverit an. In mei laboramus voluptatum. Usu cu possim impetus, pri inermis consulatu et, an affert abhorreant vis.

Usu an mediocrem dissentiunt delicatissimi. Cum dolores propriae cu. Ad sonet officiis perpetua vel, at legimus luptatum eos, pro dictas invidunt suscipiantur te. Ut movet dicit eleifend eam, vis ei graece splendide, est nihil splendide ex. Ad minim abhorreant cum, usu ut voluptua intellegat. Et his feugait epicurei, mazim nihil cu vix, nec amet dicam volumus ut.

Keywords: Flow Shop, Multi Objective, Scheduling, Ant Colony

Skripsi ini dipersembahkan untuk mamah dan papah

KATA PENGANTAR

Puji syukur penulis panjatkan kepada Tuhan Yang Maha Esa karena atas berkat dan rahmatnya penulis diberi kesabaran dan kekuatan untuk menyelesaikan penulisan skripsi ini. Penulis menyadari jika penyusunan skripsi ini tidak lepas oleh bantuan dari berbagai pihak, maka penulis ingin mengucapkan terima kasih kepada :

1. Almarhum Papah yang selalu mendoakan penulis dari kejauhan sana. Terima Kasih Pah.
2. Mamah yang selalu mendukung dan mendoakan penulis sepanjang penulisan skripsi ini.
3. Ibu Dr.rer.nat. Cecilia Esti Nugraheni yang selalu sabar dalam membimbing penuis dan memberikan masukan terhadap penulisan skripsi ini.
4. Kepada Dosen penguji yang telah memberikan masukan yang berharga untuk penulisan skripsi.
5. Staf FTIS UNPAR yang telah membantu penulis selama masa perkuliahan.
6. Kepada teman - teman angkatan 2014 yang selalu memberikan bantuan dan semangat kepada penulis.
7. Wong kar wai, David Lynch, Martin Scorsese, Stanley Kubrick, serta para director legendaris lainnya yang selalu menemani penulis selama masa penulisan skripsi dengan karya - karyanya.

Penulis berharap bahwa penulisan skripsi ini dapat membantu bagi para pembaca yang sedang meneliti atau mempelajari topik penjadwalan flow shop. Akhir kata penulis meminta maaf jika terdapat kekurangan dalam skripsi ini. Semoga penulisan skripsi ini dapat bermanfaat bagi semua pihak. Terima kasih.

Bandung, Desember 2019

Penulis

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

BAB 1

PENDAHULUAN

1.1 Latar Belakang

Penjadwalann produksi merupakan aktivitas yang tidak terpisahkan dalam suatu perusahaan *manufakturing*. Penjadwalan (*scheduling*) sendiri didefinisikan sebagai suatu proses pengalokasian sumber daya atau mesin-mesin yang ada untuk melaksanakan tugas-tugas yang ada dalam suatu waktu tertentu (Baker, 1974). Sedangkan yang dimaksud dengan proses produksi adalah serangkaian langkah-langkah yang digunakan untuk mentransformasikan *Input* menjadi *Output*.

Proses penjadwalan *Flow Shop* adalah salah satu metode penjadwalan produksi di mana urutan mesin yang digunakan untuk setiap proses dalam seluruh pekerjaan harus sama. Dalam penelitian - penelitian penjadwalan sebelumnya hanya difokuskan pada satu kriteria saja (*single*) namun pada penelitian kali ini akan menggunakan lebih dari satu kriteria (*multiple*). Banyak algoritma yang dapat digunakan untuk menentukan urutan pengerjaan pekerjaan dalam proses penjadwalan produksi *Flow Shop*. Salah satu algoritma yang dapat digunakan dalam proses penjadwalan produksi *Multi Objective Flow Shop* adalah algoritma *Ant Colony Optimization*. Algoritma *Ant Colony Optimization* adalah algoritma yang mengadopsi perilaku koloni semut yang dikenal sebagai sistem semut. Algoritma ini menyelesaikan permasalahan berdasarkan tingkah laku semut dalam sebuah koloni yang sedang mencari sumber makanan.

Penelitian ini dibuat untuk mempelajari, mengaplikasikan, serta mengukur kinerja Algoritma *Ant Colony Optimization* pada proses penjadwalan *Multi Objective Flow Shop Scheduling* (MOFSP). Pada skripsi ini juga akan dibuat perangkat lunak yang dapat menerima n job yang masing-masing terdiri atas m buah operasi dan m buah mesin. Setiap operasi hanya ditangani oleh sebuah mesin dan setiap mesin hanya bisa menangani satu operasi. Urutan operasi dari setiap job adalah sama.

1.2 Rumusan Masalah

1. Apa itu penjadwalan *Multi Objective Flowshop Scheduling* (MOFSP) ?
2. Apa itu algoritma *Ant Colony Optimization* (ACO) ?
3. Bagaimana cara kerja dan implementasi algoritma *Ant Colony Optimization* (ACO) dalam menyelesaikan permasalahan MOFSP ?
4. Bagaimana kinerja algoritma *Ant Colony Optimization* (ACO) dalam menyelesaikan permasalahan MOFSP ?

1.3 Tujuan

1. Menjelaskan penjadwalan *Multi Objective Flowshop Scheduling* (MOFSP).
2. Menjelaskan algoritma *Ant Colony Optimization* (ACO) .

3. Menampilkan cara kerja dan implementasi algoritma *Ant Colony Optimization* (ACO) dalam menyelesaikan permasalahan MOFSP .
4. Mengetahui kinerja algoritma *Ant Colony Optimization* (ACO) dalam menyelesaikan permasalahan MOFSP dengan bantuan *benchmark* tertentu.

1.4 Batasan Masalah

Batasan dan asumsi untuk penelitian ini adalah :

1. Waktu proses dari setiap pekerjaan telah diketahui dan bernilai tetap
2. Semua penyelesaian proses dari pekerjaan mengikuti alur proses yang sama dan sistematis
3. Pengerjaan proses dari pekerjaan-pekerjaan yang ada tidak dapat saling mendahului
4. Eksperimen dilakukan dengan sampel data kasus milik *Tailard Benchmark* flow shop dengan jumlah mesin yang beragam.
5. Pengukuran tingkat performansi dari algoritma menggunakan perbandingan nilai makespan dan nilai idle mesin dalam menjalankan suatu job.

1.5 Metodologi

Metodologi penyelesaian masalah dalam penelitian ini adalah :

1. Studi Literatur

   Mencari referensi dari sumber-sumber tertentu untuk memperdalam pemahaman mengenai cara kerja proses penjadwalan flow shop dan cara kerja algoritma ant colony agar kemudian dapat mengaplikasikan algoritma tersebut pada proses penjadwalan flow shop.

2. Analisa Kasus

   Menentukan cara pengaplikasian algoritma ant colony untuk optimisasi penjadwalan flow shop. Menentukan data masukan dan data keluaran yang dibutuhkan oleh perangkat lunak. Menentukan fungsi-fungsi apa saja yang dibutuhkan perangkat lunak.

3. Pengembangan perangkat lunak

   Membentuk struktur kelas dari perangkat lunak. Mendesain interface yang sesuai untuk perangkat lunak. Membangun perangkat lunak untuk optimisasi penjadwalan flow shop dengan algoritma ant colony. Melakukan pengujian fungsional pada perangkat lunak.

4. Eksperimen

   Melakukan proses optimisasi dengan menggunakan perangkat lunak pada beberapa sampel kasus flow shop. Mencatat dan mengolah data hasil proses optimisasi untuk mengukur performa perangkat lunak. Mengukur tingkat keoptimalan dari proses optimisasi yang dilakukan oleh perangkat lunak.

5. Pengambilan kesimpulan

   Mengambil kesimpulan-kesimpulan yang bisa didapatkan dari hasil eksperimen. Melakukan dokumentasi dari skripsi ini.

1.6 Sistematika Pembahasan

Sistematika penulisan karya tulis ini adalah sebagai berikut :

1. Bab 1 : Pendahuluan untuk mendefinisikan masalah yang akan dibahas, alasan pemilihan topik dan usulan solusi terhadap masalah yang ada.
2. Bab 2 : Dasar teori yang digunakan dalam penelitian ini. Pembahasan permasalahan flow shop. Pembahasan cara kerja dari algoritma ant colony. Pembahasan cara pengaplikasian algoritma ant colony untuk optimisasi proses penjadwalan flow shop.
3. Bab 3 : Analisis masalah yang akan dilakukan pada penelitian ini. Pembahasan cara penerapan algoritma ant colony dan rancangan awal dari perangkat lunak.
4. Bab 4 : Perancangan perangkat lunak. Detil informasi mengenai perangkat lunak yang telah dibuat. Struktur kelas dan desain antarmuka grafis dari perangkat lunak yang telah dibuat.
5. Bab 5 : Implementasi dan pengujian. Hasil implementasi algoritma ant colony pada perangkat lunak. Penjelasan cara penggunaan perangkat lunak. Hasil pengujian dan eksperimen kasus flow shop pada perangkat lunak
6. Bab 6 : Kesimpulan dan saran. Hal-hal yang dapat disimpulkan dari penelitian ini. Saran pengembangan yang dapat dilakukan pada penelitian selanjutnya.

BAB 2

LANDASAN TEORI

2.1 Penjadwalan Secara Umum

Secara umum penjadwalan menurut Baker (1974) didefinisikan sebagai proses pengalokasian sumber-sumber dalam jangka waktu tertentu untuk melakukan sekumpulan pekerjaan. Definisi ini mengandung dua arti yang berbeda, yaitu :

1. Penjadwalan merupakan fungsi pengambilan keputusan, yaitu menentukan jadwal.
2. Penjadwalan merupakan suatu teori, yaitu sekumpulan prinsip-prinsip dasar, model-model, teknik-teknik, dan kesimpulan-kesimpulan logis dalam proses pengambilan keputusan yang memberikan dalam fungsi penjadwalan (nilai konseptual).

Tujuan penjadwalan secara umum menurut Baker (1974) adalah :

1. Meningkatkan produktivitas mesin, yaitu dengan mengurangi waktu menganggur mesin.
2. Mengurangi terhadap persediaan barang setengah jadi, dengan mengurangi rata-rata pekerjaan yang menunggu dalam antrian karena mesin sibuk oleh pekerjaan lain.
3. Mengurangi keterlambatan (tardiness). Dalam banyak hal, beberapa atau semua pekerjaan mempunyai batas waktu penyelesaian (duedate). Apabila suatu pekerjaan melewati batas waktu tersebut, maka akan dikenai pinalti. Keterlambatan dapat diperkecil dengan mengurangi maksimal tardiness atau mengurangi pekerjaan yang terlambat (number of tardy job).

Menurut Baker [1] masalah penjadwalan muncul karena keterbatasan :

- Waktu
- Tenaga Kerja
- Jumlah Mesin
- Sifat dan syarat pekerja.

2.1.1 Istilah-Istilah dalam Penjadwalan

Bedworth [2] menyebutkan beberapa istilah dalam penjadwalan yang perlu dijelaskan adalah sebagai berikut:

- Waktu Pemrosesan (*Processing Time*)
  Lamanya waktu yang dibutuhkan untuk menyelesaikan satu aktivitas pekerjaan.
- Makespan
  Keseluruhan waktu yang dibutuhkan untuk menyelesaikan aktivitas pekerjaan.

- Waktu Aliran (Flow Time)
  Rentang waktu antara satu pekerjaan dapat dimulai sampai saat dimana pekerjaan tersebut selesai dikerjakan. Sehingga waktu aliran sama dengan waktu pemrosesan ditambah dengan waktu menunggu sebelum diproses.
- Waktu Penyelesaian (Completion Time)
  Waktu dimana tugas terakhir dari satu pekerjaan tertentu selesai dikerjakan.
- Batas Waktu (Due Time)
  Batas waktu penyelesaian untuk suatu pekerjaan (order) dan jika suatu pekerjaan (order) melewati batas waktu tersebut akan dikenakan denda (penalty)
- Tardiness
  Merupakan keterlambatan penyelesaian suatu job terhadap batas waktu penyelesaian (due time) job tersebut.

2.1.2 Klasifikasi Masalah Penjadwalan

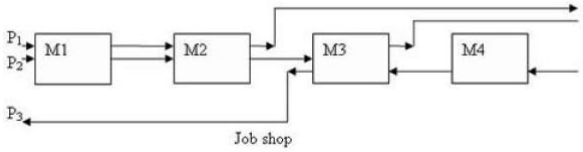
Permasalahan penjadwalan dapat dilihat dari :

1. Mesin :
   - Mesin Tunggal
   - Mesin ganda (2 mesin)
   - M mesin
2. Aliran proses
   - *Job Shop*
   - *Flow Shop*
3. Pola Kedatangan
   - Statis
   - Dinamis

2.2 Penjadwalan Flow Shop

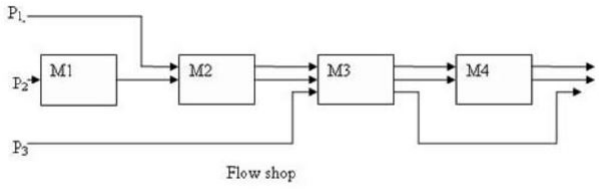
Seperti yang telah disebutkan di atas. Penjadwalan berdasarkan aliran proses terdiri atas dua penjadwalan yaitu :

1. *Job Shop*
   Setiap proses pekerjaan yang dikerjakan tidak memiliki lintasan operasi yang tetap, artinya setiap job yang datang memiliki proses operasi yang berbeda dan dapat mengalami pengulangan lintasan. Gambar 2.1 merupakan gambaran dari pola kerja job shop. Pada gambar tersebut dijelaskan bahwa P1 akan jalan melalui mesin 1 ke mesin 2. Lalu pada P3 di jelaskan job dimulai dari mesin ke 4 lalu ke mesin 3, maka dapat disimpulkan setiap job memiliki proses operasi yang berbeda.



Gambar 2.1: Job Shop

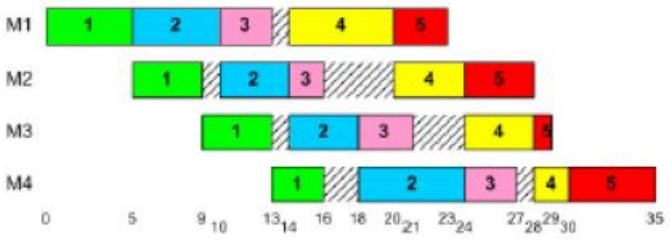
2. Flow shop
   Setiap pekerjaan yang akan diproses memiliki lintasan produksi yang searah, dan setiap operasi yang dilalui oleh setiap job memiliki urutan proses yang sama tanpa mengalami pengulangan lintasan (semua job hanya diproses sekali pada tiap mesin). Gambar 2.2 merupakan gambaran dari pola kerja flow shop.



Gambar 2.2: Flow Shop

2.2.1 Definisi Secara Umum

Permasalahan flow shop scheduling adalah bagaimana menentukan urutan pengerjaan untuk sejumlah n job dengan menggunakan sejumlah m mesin dalam urutan proses yang sama dan setiap job melewati setiap mesin dalam satu waktu tertentu. Masing-masing job memiliki lama pengerjaan yang berbeda. Dalam pemrosesan, setiap job membutuhkan sebanyak m operasi (setiap mesin satu operasi). Setiap job tidak dapat saling mendahului dan setiap mesin tidak akan diam saat job siap diproses. Tujuan utama dari flow shop scheduling adalah menentukan makespan minimum untuk mengerjakan n job dengan menggunakan m mesin. Nilai makespan dapat berubah-ubah berdasarkan urutan pengerjaan job yang dilakukan, oleh karena itu urutan pengerjaan akan sangat berpengaruh terhadap nilai makespan. Pada skripsi ini juga akan ditambahkan satu *objective* lainnya dalam mengukur performa algoritma *ant colony* terhadap permasalahan flow shop. Objective tersebut adalah meminimasi terjadinya tardiness dalam suatu penyelesaian jobİlustrasi dari Flow Shop Scheduling dapat dilihat pada gambar 2.3 berikut :



Gambar 2.3: Ilustrasi Flow Shop

Gambar 2.3 merupakan ilustrasi proses flowshop scheduling. Pada gambar tersebut urutan pengerjaan job adalah 1,2,3,4, dan 5. Masing-masing job memiliki 4 proses operasi dan masing-masing proses operasi tersebut dikerjakan oleh mesin yang berbeda. Saat proses operasi 1 dari job 1 telah selesai, mesin 2 akan mengerjakan proses operasi 2 dari job 1, dan mesin 1 akan mengerjakan proses operasi 1 dari job 2. Saat proses operasi 2 dari job 1 sudah selesai, mesin 2 akan mengerjakan proses operasi 3 dari job 1, akan tetapi mesin 2 yang seharusnya mengerjakan proses operasi 2 dari job 2 akan mengalami idle, karena proses operasi 1 dari job 2 belum selesai dikerjakan di mesin 1.

Idle adalah kondisi dimana mesin tidak melakukan proses operasi apapun, baik itu karena adanya proses operasi yang akan dilakukan pada mesin tersebut belum selesai diproses di mesin selanjutnya atau karena masih ada proses operasi yang dilakukan pada mesin selanjutnya. Idle

akan mempengaruhi besar kecilnya nilai makespan yang akan dihasilkan. Semakin banyak idle, maka makespan umumnya akan semakin besar, serta sebaliknya dimana semakin sedikit idle maka umumnya makespan yang didapat akan semakin kecil. Mesin juga mengalami idle pada saat mesin 1 telah menyelesaikan proses operasi job 3 namun ternyata pada mesin 2 proses job 2 belum selesai. Sehingga job 3 akan disimpan dalam mesin 1 hingga mesin 2 selesai mengerjakan proses operasi job 2.

2.2.2 Karakteristik Penjadwalan Flow Shop

1. Terdapat n job yang tersedia dan siap diproses pada waktu t = 0.
2. Waktu proses tiap job tidak bergantung pada urutan pengerjaan.
3. Terdapat m mesin berbeda, yang tersedia secara kontinu.
4. Operasi-operasi individual job pada tiap mesin tidak dapat dipecah-pecah.

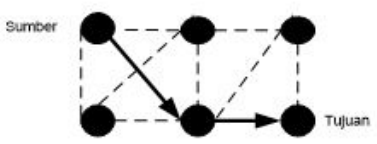
2.3 Multi Objective Flow Shop Scheduling

Pada dasarnya MOFSP adalah bagian dari *flow shop* itu sendiri. MOFSP menjelaskan kriteria optimasi yang akan diselesaikan menggunakan algoritma *Ant Colony Optimization*. Kriteria tersebut bersifat jamak *Multi Objective*. Kriteria tersebut misalnya saja minimasi waktu proses keseluruhan, minimasi waktu tunggu job, minimasi waktu nganggur mesin, dan sebagainya. Kriteria optimasi yang akan diselesaikan pada skripsi ini adalah menentukan makespan minimum dan minimasi waktu *idle* atau *tardiness*.

2.4 Ant Colony Optimization

Any Colony Optimization (ACO) awalnya dikembangkan oleh Marco Dorigo et. al. (1996). Algoritma ini didasarkan pada cara kerja semut untuk menentukan jarak terpendek dari sarang menuju sumber makanan. Semut dapat menemukan jarak terpendek dengan memanfaatkan jejak pheromone (air liur semut) yang dimanfaatkan sebagai komunikasi tidak langsung antar semut. Ketika semut berjalan, ia meninggalkan pheromone dalam jumlah tertentu pada jalur yang dilewatinya. Semut dapat mencium pheromone dan ketika memilih jalur mereka cenderung untuk memilih jalur dengan konsentrasi pheromone yang lebih besar adalah jarak terpendek.

Saat seekor semut yang terisolasi bergerak secara acak, semut ini akan mengikuti jejak yang telah ditinggalkan sebelumnya yang dapat dideteksi dan mempunyai tingkat probabilitas yang tinggi untuk diikuti dan melanjutkan jejak sebelumnya dengan pheromone baru. Tingkah laku kolektif yang muncul disebut dengan tingkah laku Autocalystic, dimana semut yang lain dapat mengikuti jejak yang ada dan jejak yang semakin jelas akan memudahkan bagi semut yang lain untuk mengikutinya. Proses ini secara khusus terjadi melalui kumpulan umpan balik yang positif, dimana kemungkinan semut untuk memilih pola meningkat seiring dengan jumlah semut yang sebelumnya mengikuti pola yang sama.



Gambar 2.4: Jalur Solusi Semut Ant Colony Optimization

Marco Dorigo et. al. (1996) mengatakan bahwa Ant Colony Optimization adalah algoritma heuristik yang serba guna untuk memecahkan berbagai masalah optimasi. Algoritma ACO memiliki karakteristik sebagai berikut :

1. Serba guna (versatile), dapat dipakai untuk memecahkan masalah dengan versi yang sama, seperti TSP dan Asymetric Travelling Salesman Problem (ATSP).

2. Sempurna (robust), dapat diterapkan untuk memecahkan dengan hanya perubahan sedikit terhadap masalah optimasi yang lain, seperti Quadratic Assigment Problem dan Job shop Scheduling Problem (JSP).

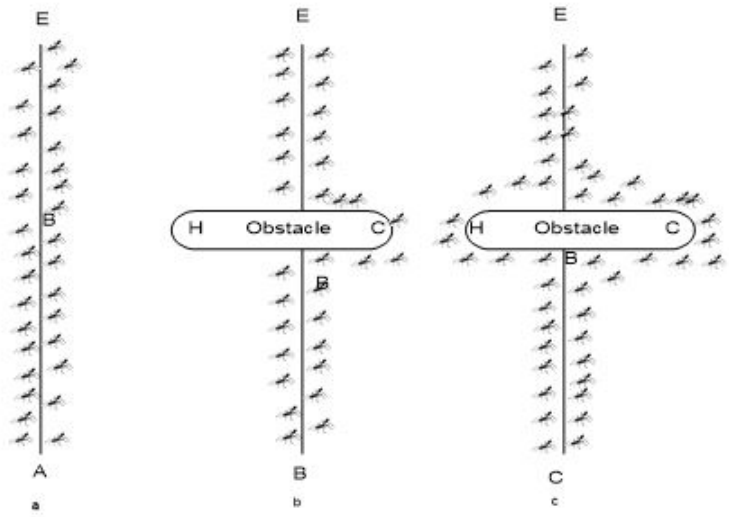
3. Pendekatan yang berbasis populasi.

2.4.1 Algoritma Ant Colony Optimization pada penjadwalan Flow Shop

- Karakteristik Semut
  Sesuai dengan contoh pada gambar 2.5 terdapat pola saat sekelompok semut berjalan (contoh, dari sumber makanan A ke sarang E, dan sebaliknya). Secara tiba-tiba rintangan muncul dan pola menjadi terpotong. Pada posisi B, semut berjalan dari A ke E (atau pada posisi D yang bergerak dengan arah yang berlawanan) harus memutuskan apakah harus bergerak ke kiri atau ke kanan. Pilihan ini dipengaruhi oleh intensitas jejak pheromone yang ditinggalkan oleh semut sebelumnya. Tingkat pheromone yang lebih tinggi pada pola sebelah kanan memberikan semut rangsangan yang lebih kuat dan kemungkinan yang lebih tinggi untuk berbelok ke kanan. Semut pertama mencapai titik B atau D mempunyai kemungkinan yang sama untuk belok ke kiri atau ke kanan (karena tidak terdapat pheromone pada dua pola alternatif tersebut).

  Karena pola BCD lebih pendek dibandingkan dengan pola BHD, semut pertama yang mengikuti ini akan mencapai D sebelum semut yang mengikuti pola BHD. Hasilnya adalah semut yang bergerak dari E ke D akan mendapatkan jejak yang lebih jelas pada pola DCB, karena setengah dari semut tersebut yang memilih untuk mendekati rintangan melalui DCBA dan dengan segera akan sampai melalui BCD, mereka akan melalui pola memilih pola DCB dibandingkan pola DHB.

  Sebagai Konsekuensi, jumlah semut yang mengikuti pola BCD per unit waktu akan lebih banyak dibandingkan dengan jumlah semut yang mengikuti pola BHD. Hal ini menyebabkan jumlah pheromone pada pola yang lebih pendek akan muncul lebih cepat dibandingkan dengan pola yang lebih jauh, dan oleh karena itu kemungkinan semut yang memilih pola yang diikuti mempunyai bias terhadap pola yang lebih pendek. Hasil akhir yang akan dipilih secara cepat akan ditunjukkan pada pola yang lebih pendek. Algoritma yang akan dibahas pada bagian selanjutnya adalah model yang berasal dari kumpulan kehidupan nyata semut. Selanjutnya hal ini disebut dengan Sistem Semut dan algoritma yang akan dibahas dikenal dengan algoritma semut. Karakteristik semut asli untuk ketika sedang bergerak dari satu titik ke titik tujuan dapat dilihat pada kedua ilustrasi gambar berikut ini.

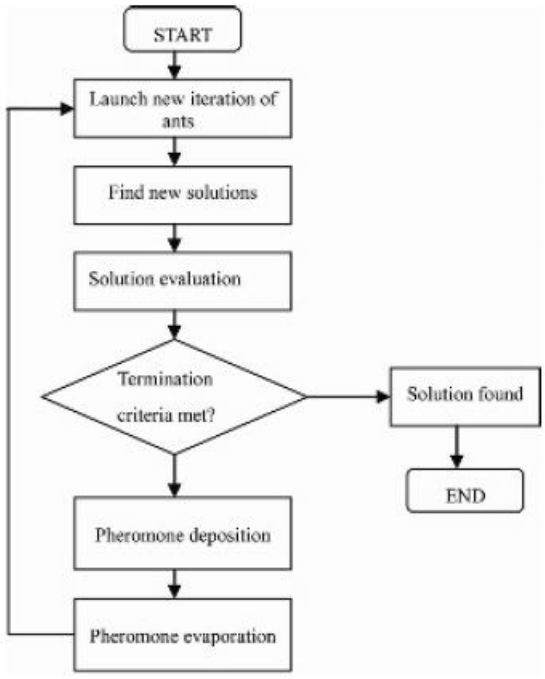


Gambar 2.5: Karakteristik semut

Penjelasan karakteristik semut :

1. Semut-semut mengikuti jalur dari titik A ke titik E.
2. Sebagai rintangan maka diletakkan hambatan pada jalur lintasan ; semut-semut memilih untuk bergerak memutari salah satu sisi dengan probabilitas yang sama.
3. Di jalur yang lebih pendek ditinggalkan lebih banyak pheromone.

- Cara Kerja Algoritma Berikut adalah flow chart dari algoritma ant colony.



Gambar 2.6: Flowchart algoritma ant colony optimization

Berdasarkan flowchart 2.6, dalam menjalankan proses pelatihan dengan algoritma ant colony maka perlu diperrsiapkan jalur-jalur yang ada perlu. Jalur-jalur tersebut merupakan kemungkinan solusi-solusi yang mungkin dipilih oleh semut. Kemungkinan dipilihnya suatu jalur akan ditentukan berdasarkan nilai feromon, oleh karena itu perlu dibentuk pula tempat penyimpanan feromon untuk setiap jalur yang mungkin dilalui.

Pada awalnya, algoritma ant colony akan membentuk sekumpulan semut, semut-semut tersebut akan disebar secara acak pada jalur-jalur yang ada. Proses penyebaran semut tersebut akan memperhatikan nilai feromon dari masing-masing jalur. Semakin besar nilai feromon dari suatu jalur, maka semakin besar pula kemungkinan dipilihnya jalur tersebut untuk dilalui semut. Jalur-jalur yang dipilih oleh semut akan disimpan dan dibandingkan dengan jalur-jalur yang telah dipilih oleh semut-semut lainnya.

Dari jalur-jalur yang telah dipilih akan dibentuk nilai feromonnya berdasarkan tingkat keoptimalan dari jalur tersebut. Nilai feromon-feromon tersebut akan ditambahkan pada jalur-jalur yang bersangkutan untuk memperbesar kemungkinan dipilihnya jalur tersebut sesuai dengan tingkat keoptimalannya. Semakin optimal suatu jalur, semakin besar pula jumlah feromon yang akan ditambahkan pada suatu jalur. Tidak lupa pula, proses evaporasi feromon akan dilakukan. Proses evaporasi feromon ini akan mengurangi jumlah feromon yang disimpan. Proses evaporasi ini bertujuan untuk mengurangi jumlah feromon dari jalur-jalur yang dianggap kurang optimal. Proses-proses tersebut akan terus dilakukan hingga suatu kondisi berhenti ditemui.

```
membentukDataFeromon()
solusiTerbaik = null
while(kondisi berhenti belum ditemui)
    koloniSemut = bentuk kumpulan semut kosong
    for(1 sampai jumlah semut yang akan disebar)
        solusiLokal = pilihSolusiSecaraAcakBerdasarkanNilaiFeromon()
        If(solusiLokal solusi yang valid)
            (opsional) localSearch(solusiLokal)
            if(hasil(solusiLokal) lebih baik dari hasil(solusiTerbaik))
                simpan solusiLokal pada solusiTerbaik
            end if
            simpan solusiLokal pada koloniSemut
        end if
    end for
    updateNilaiFeromon(koloniSemut)
end while
output : solusiTerbaik
```

Gambar 2.7: Pseudocode algoritma ant colony optimization

Dari flow chart 2.6, dibentuklah pseudocode di atas. Baris pertama dari pseudocode tersebut merupakan proses pembentukan data feromon. Proses pembentukan data feromon tersebut akan memperhatikan kasus yang ingin dicari hasil optimalnya. Nilai masingmasing jalur dan cara mengartikan nilai feromon tersebut diatur berdasarkan kasus yang ingin dicari hasil optimalnya. Jumlah feromon pada masing-masing jalur pada awalnya bernilai sama. Setelah data feromon selesai dibuat, proses pelatihan dengan menggunakan algoritma ant colony dimulai. Baris ketiga pada pseudocode menunjukkan kondisi berhenti untuk proses optimisasi dari algoritma ant colony.

Seluruh potongan kode di dalam kondisi while tersebut merupakan satu fase pelatihan dari algoritma ant colony. Fase pelatihan tersebut terdiri dari proses pembentukan jalur, penentuan solusi optimal, dan perubahan nilai feromon (penambahan dan evaporasi feromon). Fase pelatihan dari algoritma ant colony bertujuan untuk membentuk data feromon yang mampu

menghasilkan solusi yang paling optimal. Pada setiap fase pelatihan ini akan dibentuk sejumlah semut yang bertugas untuk menyimpan solusi acak/solusi lokal yang telah dipilih pada suatu fase pelatihan. Semut-semut ini akan dianggap memilih sebuah jalur/solusi secara acak berdasarkan nilai yang disimpan pada data feromon. Jika solusi acak tersebut lebih baik dari solusi optimal yang diketahui pada saat ini, maka solusi acak tersebut akan dianggap sebagai solusi optimal yang baru.

Pada saat pemilihan solusi secara acak, perlu dipastikan bahwa solusi tersebut merupakan solusi yang valid dari suatu kasus. Pada saat pemilihan solusi kita juga dapat melakukan proses local search. Proses ini bertugas untuk membandingkan solusi yang dibentuk dengan solusi yang mirip. Jika solusi lain tersebut lebih baik, maka solusi yang dipilih akan diganti menjadi solusi yang mirip tersebut. Proses ini bersifat opsional, karena tidak semua permasalahan dapat diaplikasikan dengan proses ini.

Setelah semut-semut telah selesai memilih solusi secara acak, proses perubahan nilai feromon akan dilakukan. Proses perubahan tersebut mencakup proses penambahan dan evaporasi nilai feromon. Proses penambahan nilai feromon akan memperhatikan tingkat keoptimalan dari solusi yang dipilih suatu semut. Semakin optimal suatu solusi, semakin banyak pula nilai feromon yang ditambahkan pada jalur/solusi tersebut. Setelah proses optimisasi selesai, solusi yang paling optimal akan diberikan sebagai data keluaran.

2.4.2 Rumus - rumus Algoritma Ant Colony dalam penjadwalan Flow Shop

1. Pemilihan Jalur
   Kemungkinan dipilihnya sebuah jalur dipengaruhi oleh kekuatan feromon pada jalur tersebut. Semakin kuat / banyak feromon pada suatu jalur, maka jalur tersebut akan lebih sering / berkemungkinan lebih besar untuk dipilih. Kemungkinan terpilihnya masing-masing jalur dapat dihitung dengan menggunakan rumus yang terdapat pada rumus berikut :

$$P_k^{xy} = \frac{T_{xy}^{\alpha} \cdot n_{xy}^{\beta}}{\sum_{x \in availeble_z} (T_{xz}^{\alpha} \cdot n_{xz}^{\beta})} \tag{2.1}$$

Keterangan dari rumus tersebut sebgai berikut :

P_k^{xy} : nilai kemungkinan terpilihnya jalur xy oleh semut k
T_{xy} : nilai feromon untuk jalur xy
n_{xy} : nilai objektif untuk jalur xy
α : persentase pengaruh nilai feromon
β : persentase pengaruh nilai objektif

T menunjukkan kekuatan / jumlah feromon yang terdapat pada suatu jalur, sedangkan n menunjukkan nilai objektif yang dimiliki oleh suatu jalur. Nilai objektif pada suatu jalur menunjukkan pengetahuan / nilai awal mengenai jalur tersebut. Nilai awal tersebut biasanya berupa informasi mengenai jalur tersebut yang diketahui sejak awal dan mampu memberikan pengaruh pada proses pemilihan jalur.

Contoh informasi yang dapat menjadi nilai objektif adalah panjang jalur, banyaknya belokan, panjang jalur pertama, dan lain-lain. Nilai awal suatu jalur dapat bernilai sama dengan jalur yang lain. Apabila proses pencarian yang dilakukan tidak memiliki informasi apapun atau bersifat blind- search, nilai objektif ini dapat diabaikan. Besar pengaruh dari kekuatan feromon dan nilai objektif terhadap kemungkinan terpilihnya sebuah jalur dapat diatur dengan menggunakan rasio tertentu.

2. Update Nilai Feromon
   Ketika terdapat semut yang menyelesaikan sebuah jalur, feromon pada jalur tersebut akan diubah. Jumlah feromon pada jalur yang dilewati semut tersebut akan ditambah. Dalam proses penambahan feromon, nilai feromon yang baru pada jalur ditentukan dengan rumus berikut:

$$T_{xy} \leftarrow (1 - P) \cdot T_{xy} + \sum_{k} \Delta T_{xy}^{k} \tag{2.2}$$

   Keterangan dari rumus tersebut sebgai berikut :

   P : persentase evaporasi feromon (dalam desimal)
   T_{xy} : nilai feromon untuk jalur xy
   ΔT_{xy}^{k} : nilai feromon yang akan ditambahkan oleh semut k pada jalur xy

   Pada rumus tersebut, jumlah feromon awal yang terdapat pada suatu jalur akan mengalami pengurangan berdasarkan rasio penguapan feromon P, lalu kemudian akan ditambah dengan nilai feromon masing-masing semut yang melewati jalur tersebut. Nilai feromon masing-masing semut dapat berupa sebuah nilai tetap atau nilai lainnya yang mampu memberikan nilai lebih pada jalur yang lebih cepat (lama proses, jumlah semut yang telah melewati suatu jalur, dan lain-lain). Aturan mengenai kapan dan bagaimana suatu proses penambahan feromon dilakukan dapat bervariasi. Aturan-aturan tersebut biasanya ditentukan berdasarkan jenis permasalahan yang ingin dioptimisasi. Proses penambahan feromon dapat dilakukan ketika terdapat semut yang telah selesai berjalan pada jalurnya atau pada saat seluruh semut telah selesai berjalan pada jalurnya.

   Proses perubahan jumlah feromon dapat dilakukan dengan menambahkan sebuah nilai atau penambahkan nilai feromon dengan suatu persentase tertentu. Proses perubahan jumlah feromon dapat dilakukan di seluruh jalur atau hanya di jalur yang dilewati suatu semut. Jika perubahan dilakukan di seluruh jalur, fungsi / aturan khusus untuk perubahan jumlah feromon sebagai berikut.

$$\Delta T_{xy}^{k} = \{\, Q \text{ , jika semut } \mathbf{k} \text{ menggunakan jalur } \mathbf{xy}.0, \text{untuk lainnya.} \tag{2.3}$$

   Keterangam aturan khusus di atas sebagai berikut :

   ΔT_{xy}^{k} : nilai feromon yang akan ditambahkan oleh semut k pada jalur xy
   Q : nilai penambahan feromon (dapat berupa suatu angka tetap / hasil perhitungan)

3. Kondisi berhenti
   Selama proses optimisasi, algoritma ant colony akan secara terus menerus melakukan fase pelatihan. Pada setiap fase pelatihan akan dibentuk semut-semut yang akan memilih suatu jalur secara acak berdasarkan nilai feromon. Setelah suatu fase pelatihan selesai, perubahan nilai feromon akan dilakukan berdasarkan jalur-jalur yang dipilih oleh masing-masing semut.

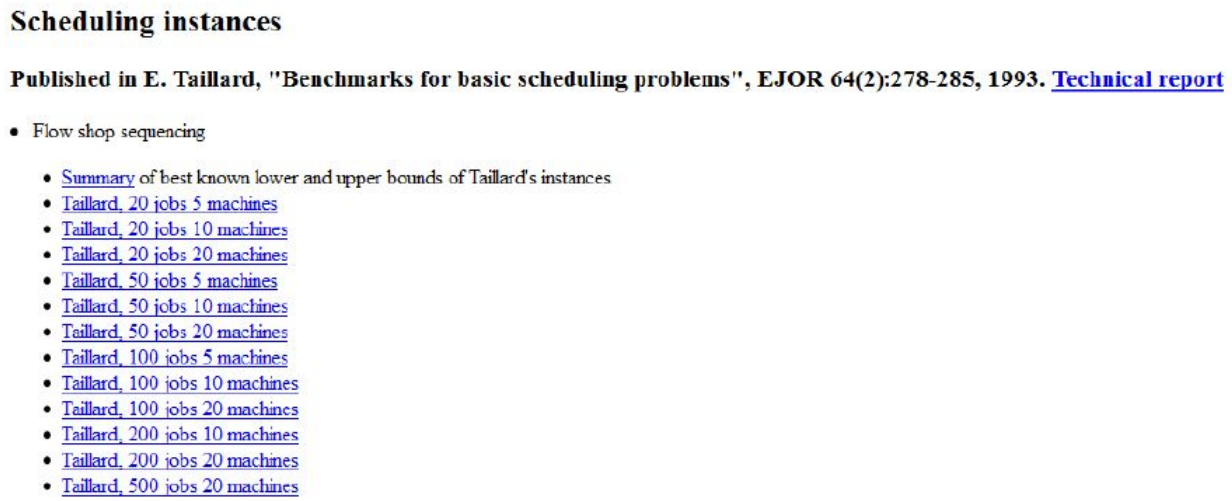
   Proses optimisasi akan diberhentikan jika suatu kondisi telah terpenuhi. Waktu berhentinya suatu proses pelatihan dapat ditentukan dengan berbagai parameter. Proses pelatihan dapat dianggap selesai jika telah melewati beberapa fase pelatihan, jumlah semut yang disebar sudah cukup banyak, atau jika sudah terdapat jalur yang dipilih oleh sebagian besar semut. Proses pelatihan juga dapat diberhentikan jika fase-fase pelatihan terakhir selalu memberikan hasil solusi yang sama / tidak mampu membentuk solusi lain yang lebih optimal.

2.5 Tailard Benchmark

Dalam melakukan pengujian tingkat keoptimalan dari suatu algoritma, pengujian sebaiknya dilakukan dengan menggunakan data kasus standar. Data kasus standar tersebut akan dianggap sebagai suatu kasus unik yang memerlukan algoritma khusus untuk proses optimisasinya. Data standar tersebut akan digunakan sebagai pembanding dalam proses optimisasi. Dengan menggunakan data standar, penilaian kualitas dari suatu algoritma optimisasi dapat dilakukan. Dalam kasus penelitian ini, data kasus standar tersebut akan menggunakan data dari suatu benchmark yang bernama *Taillard Benchmark*. *Benchmark* tersebut dapat diakses lewat url berikut : [1] .

Taillard benchmark adalah sebuah web hosting yang menyediakan kumpulan data yang digunakan sebagai input dan akan menghasilkan output suatu permasalahan flow shop scheduling. Input dan output tersebut digunakan sebagai standar dalam pengujian algoritma penyelesaian flow shop scheduling sesuai dengan yang didefinisikan oleh Taillard [9]. Hal ini akan mempengaruhi penilaian suatu algoritma dalam menyelesaikan permasalahan flow shop scheduling apakah algoritma tersebut sudah efektif atau tidak.

Dalam data input yang digunakan pada permasalahan flow shop scheduling, selain banyaknya job dan mesin, ada pula initial seed yang dapat digunakan menggunakan random generator. Initial Seed ini berfungsi dalam membuat variasi waktu proses dari tiap job yang ada, sehingga kumpulan job yang ada akan memiliki waktu proses yang berbeda pada tiap prosesnya. Seperti yang telah diketahui sebelumnya, tujuan utama dari pencarian solusi permasalahan flow shop scheduling adalah pencarian waktu makespan yang paling kecil. Taillard benchmark Framework ini akan menyediakan data output yang berupa upper bound dan lower bound solusi permasalah flow shop scheduling. Lower bound menunjukkan batas makespan paling minimum sedangkan Upper bound menunjukkan batas makespan paling maksimum. Selain itu, Taillard benchmark juga akan memberikan data-data detail dari proses-proses yang dilakukan.Tampilan dari Tailard benchmark sebagai berikut :



Gambar 2.8: Tailard Benchmark

Gambar 2.8 memperlihatkan tampilan menu Taillard benchmark yang dimana kita dapat memilih kasus-kasus flow shop scheduling yang hendak digunakan. Pada setiap kasusnya, telah disediakan data-data input berupa jumlah job dan jumlah mesin dan initial seed. Selain itu diberikan pula upper bound dan lower bound dari kasus tersebut untuk menjadi tolak ukur algoritma yang telah kita buat apakah dapat memenuhi standar dari Taillard Benchmark. Taillard membagi jenis kasus-kasus flow shop scheduling berdasarkan jumlah job dan jumlah mesin. Pada tiap jenis kasus Taillard menyediakan 10 (sepuluh) kasus yang dapat dicoba.

[1] http://mistic.heig-vd.ch/taillard/problemes.dir/ordonnancement.dir/ordonnancement.html

```
number of jobs, number of machines, initial seed, upper bound and lower bound :
         20           5   873654221        1278        1232
processing times :
 54 83 15 71 77 36 53 38 27 87 76 91 14 29 12 77 32 87 68 94
 79  3 11 99 56 70 99 60  5 56  3 61 73 75 47 14 21 86  5 77
 16 89 49 15 89 45 60 23 57 64  7  1 63 41 63 47 26 75 77 40
 66 58 31 68 78 91 13 59 49 85 85  9 39 41 56 40 54 77 51 31
 58 56 20 85 53 35 53 41 69 13 86 72  8 49 47 87 58 18 68 28
number of jobs, number of machines, initial seed, upper bound and lower bound :
         20           5   379008056        1359        1290
processing times :
 26 38 27 88 95 55 54 63 23 45 86 43 43 40 37 54 35 59 43 50
 59 62 44 10 23 64 47 68 54  9 30 31 92  7 14 95 76 82 91 37
 78 90 64 49 47 20 61 93 36 47 70 54 87 13 40 34 55 13 11  5
 88 54 47 83 84  9 30 11 92 63 62 75 48 23 85 23  4 31 13 98
 69 30 61 35 53 98 94 33 77 31 54 71 78  9 79 51 76 56 80 72
number of jobs, number of machines, initial seed, upper bound and lower bound :
         20           5  1866992158        1081        1073
processing times :
 77 94  9 57 29 79 55 73 65 86 25 39 76 24 38  5 91 29 22 27
 39 31 46 18 93 58 85 58 97 10 79 93  2 87 17 18 10 50  8 26
 14 21 15 10 85 46 42 18 36  2 44 89  6  3  1 43 81 57 76 59
 11  2 36 30 89 10 88 22 31  9 43 91 26  3 75 99 63 83 70 84
 83 13 84 46 20 33 74 42 33 71 32 48 42 99  7 54  8 73 30 75
number of jobs, number of machines, initial seed, upper bound and lower bound :
         20           5   216771124        1293        1268
```

Gambar 2.9: Contoh kasus Tailard Benchmark

Gambar 2.9 memperlihatkan tampilan salah satu kasus pada flow shop scheduling dengan 20 jobs dan 5 mesin. Pada gambar tersebut kita dapat melihat data waktu proses tiap job pada tiap mesin. Waktu proses ini didapatkan seluruhnya berdasarkan initial seed yang digunakan. Berdasarkan data tersebut, algoritma yang telah dibuat sebelumnya dapat dinilai apakah telah efektif atau belum sesuai dengan upper bound dan lower bound yang diberikan.

BAB 3

ANALISA MASALAH

Pada bab ini akan dibahas mengenai hasil analisa permasalahan flow shop dan segalah hal yang berkaitan dengan analisa permasalahan tersebut. Pada bab ini juga akan dibahas sekilas mengenai rancangan awal dari perangkat lunak yang memungkinkan diaplikasikannya algoritma ant colony pada permasalahan multi objective flow shop.

3.1 Analisa Kasus

Proses penjadwalan Flow Shop adalah salah satu metode penjadwalan produksi di mana urutan mesin yang digunakan untuk setiap proses dalam seluruh pekerjaan harus sama. Pekerjaan-pekerjaan tersebut akan dikerjakan pada mesin-mesin yang ada dengan urutan pengerjaan yang dipilih. Masing - masing proses akan dikerjakan secara sistematis dan terurut. Data-data masukan yang dibutuhkan dalam menjalankan sebuah proses penjadwalan flow shop adalah:

- Banyaknya pekerjaan
- Banyaknya mesin
- Detail waktu pengerjaan setiap pekerjaan

Setiap pekerjaan memiliki waktu proses yang berbeda satu sama lain pada tiap mesin. Variasi urutan pengerjaan yang berbeda biasanya akan menghasilkan waktu pengerjaan / makespan yang berbeda pula. Proses penjadwalan ini penting untuk diperhatikan, karena mampu mempengaruhi lama waktu penggunaan fasilitas dalam suatu proses produksi. Urutan pengerjaan yang baik dapat dicari dengan menggunakan suatu algoritma optimisasi. Diharapkan dengan algoritma optimisasi tersebut, urutan pengerjaan yang menghasilkan makespan / waktu pengerjaan minimum dapat ditemukan.

Salah satu algoritma yang dapat digunakan untuk optimisasi penjadwalan flow shop adalah algoritma ant colony. Tingkat keoptimalan setiap algoritma berbeda-beda, oleh karena itu perlu dibuat sebuah perangkat lunak untuk optimisasi penjadwalan flow shop dengan menggunakan algoritma ant colony. Algoritma ant colony akan menerima data masukan dari suatu kasus flow shop, kemudian akan mencari urutan pengerjaan / solusi yang paling optimal dari kasus tersebut selain itu kita juga dapat mencari kriteria lain yang kita inginkan.

Algoritma ant colony akan mencari urutan pengerjaan yang menghasilkan makespan / waktu pengerjaan yang paling minimum. Algoritma ant colony merepresentasikan pilihan solusi-solusi yang ada sebagai jalur yang akan dilalui oleh semut dan memberikan data feromon sebagai panduan untuk melewati jalur-jalur tersebut. Dengan panduan tersebut, semut-semut tersebut diasumsikan mampu memilih jalur yang paling optimal. Proses optimisasi dengan menggunakan algoritma ant colony perlu memperhatikan beberapa hal.

Algoritma ant colony perlu mengetahui bagaimana cara merepresentasikan suatu solusi sebagai jalur, bagaimana cara menggunakan panduan feromon untuk pemilihan jalur, bagaimana cara membaca dan menyimpan suatu data feromon, serta banyak hal lainnya. Tata cara tersebut mampu mempengaruhi hasil optimisasi dari algoritma ant colony.

3.2 Analisa Algoritma Ant Colony Pada Penjadwalan Flow Shop

3.2.1 Urutan Pengerjaan Proses Sebagai Jalur Semut

Algoritma ant colony akan membantu penentuan urutan pengambilan pekerjaan yang optimal. Oleh karena itu, algoritma ini akan merepresentasikan pilihan urutan pengambilan yang ada sebagai jalur. Jalur-jalur tersebut kemudian akan dilewati oleh semut-semut yang akan mencari solusi / urutan pengerjaan yang optimal. Pilihan urutan pengerjaan yang ada dapat dicari dengan menggunakan fungsi permutasi terhadap jumlah pekerjaan.

Sebagai contoh, jika dimisalkan terdapat 3 buah pekerjaan, maka dengan fungsi permutasi akan didapatkan jalur-jalur urutan pengerjaan (1,2,3), (1,3,2), (2,1,3), (2,3,1), (3,1,2), dan (3,2,1). Fungsi permutasi ini digunakan dengan mempertimbangkan sifat dari proses penjadwalan flow shop. Masing-masing proses dari pekerjaan pasti akan dikerjakan dan hanya akan dikerjakan sebanyak satu kali.

3.2.2 Memilih Urutan Pengerjaan / Jalur

Makespan yang dihasilkan oleh masing-masing urutan pengerjaan biasanya akan bernilai berbeda. Penentuan urutan pengerjaan yang optimal akan ditentukan oleh nilai feromon yang akan diberikan oleh semut-semut yang telah disebar. Nilai feromon tersebut dapat disimpan dengan cara yang beragam.

Dalam kasus flow shop juga terdapat beragam cara dalam menyimpan feromon dan penentuan urutan yang lebih optimal. Salah satunya adalah dengan menyimpan nilai feromon sebagai panduan pemilihan pekerjaan selanjutnya. Nilai feromon akan menyimpan kecenderungan dipilihnya suatu pekerjaan setelah dipilihnya suatu pekerjaan yang lain. Nilai feromon tersebut akan disimpan dalam bentuk matriks 2 dimensi.

Indeks pertama dari matriks akan merepresentasikan nomor pekerjaan yang sebelumnya dipilih. Indeks kedua akan merepresentasikan pekerjaan yang selanjutnya akan dipilih. Sebagai contoh, pada matriks indeks [1][2] menunjukkan kecenderungan dipilihnya pekerjaan 2 setelah dipilihnya pekerjaan 1. Nilai feromon pada matriks dengan 2 indeks yang sama akan selalu bernilai 0.

Perlu diingat pula bahwa algoritma ant colony akan selalu memberikan peluang dipilihnya sebuah solusi. Meskipun solusi tersebut sudah dapat dianggap sebagai solusi yang kurang optimal, solusi tersebut harus tetap memiliki kemungkinan untuk dipilih. Oleh karena itu, indeks nilai feromon harus tetap dapat mengacu pada terpilihnya suatu solusi-solusi yang ada. Nilai feromon masing-masing indeks matriks (kecuali matriks dengan dua indeks yang sama) tidak akan lebih kecil dari 1. Hal ini dilakukan agar setiap solusi masih memiliki kemungkinan untuk dipilih, meskipun kemungkinannya sangat kecil.

3.2.3 Proses Pemilihan Urutan Pengerjaan / Jalur Semut

Pada proses pemilihan jalur, nilai feromon yang disimpan oleh suatu indeks matriks mempengaruhi kemungkinan dipilihnya suatu pekerjaan. Semakin besar nilai feromon pada suatu indeks, semakin besar kemungkinan dipilihnya pekerjaan yang dirujuk oleh indeks matriks tersebut. Berdasarkan rumus untuk pemilihan jalur pada algoritma ant colony, diperlukan nilai objektif dan nilai feromon untuk penentuan pemilihan suatu jalur. Pada proses pemilihan pekerjaan ini, nilai objektif yang digunakan adalah 1 untuk semua indeks matriks. Proses pemilihan pekerjaan ini hanya akan dipengaruhi oleh nilai feromon yang disimpan pada indeks matriks yang ada.

Penentuan pekerjaan mana yang akan dipilih dapat dimulai dengan melakukan proses randomisasi suatu angka. Angka tersebut akan berada di kisaran nilai 1 hingga jumlah nilai feromon dari indeks-indeks yang mungkin dilibatkan. Penentuan pekerjaan dilakukan dengan menambahkan secara satu per satu nilai feromon dari indeks yang terlibat dimulai dari nilai 0. Jika setelah ditambahkan nilai dari suatu indeks, jumlah nilai telah melebihi angka hasil randomisasi, maka pekerjaan yang dirujuk indeks tersebut akan dipilih sebagai pekerjaan yang diambil.

Proses pemilihan jalur dilakukan secara bertahap, dimulai dari pemilihan jalur pertama. Untuk pemilihan jalur pertama, nilai feromon yang mempengaruhi terpilihnya suatu jalur merupakan jumlah nilai feromon pada matriks dengan indeks kedua yang merujuk pada jalur tersebut. Sebagai contoh, jika dimisalkan terdapat 2 buah pekerjaan, maka jumlah nilai matriks pada indeks [1][1] dan indeks [2][1] merupakan nilai feromon untuk terpilihnya pekerjaan 1 untuk dikerjakan pertama kali.

Untuk pemilihan pekerjaan selanjutnya dilakukan dengan representasi normal dari matriks tersebut. Indeks pertama berfungsi sebagai penunjuk pekerjaan yang diambil sebelumnya dan indeks kedua merupakan penunjuk pekerjaan yang akan diambil selanjutnya. Proses pemilihan ini akan memperhatikan pekerjaan-pekerjaan yang sebelumnya telah dipilih. Sebagai contoh, jika dimisalkan terdapat 4 buah pekerjaan dan urutan pekerjaan yang sudah dipilih adalah [1,2], maka indeks matriks yang akan diperhatikan dalam proses pemilihan hanya indeks matriks [2][3] dan [2][4]. Indeks matriks [2][1] tidak diperhatikan karena sudah diambil untuk dikerjakan pertama kali.

3.2.4 Proses Penambahan Feromon

Setiap solusi yang dicoba akan menghasilkan makespan yang berbeda-beda. Nilai makespan yang dihasilkan mempengaruhi tingkat keoptimalan dari solusi tersebut. Semakin kecil nilai makespan yang dihasilkan, semakin besar tingkat keoptimalan dari suatu solusi.

Nilai feromon yang disimpan harus mampu mempengaruhi semut-semut untuk lebih sering memilih jalur yang lebih optimal. Oleh karena itu, nilai feromon pada solusi optimal harus lebih cepat bertambah. Hal tersebut dapat dilakukan dengan memanipulasi banyak feromon yang ditambahkan. Semakin kecil nilai makespan yang dihasilkan, semakin besar nilai feromon yang akan disimpan atau ditambahkan.

Nilai feromon yang akan ditambahkan merupakan hasil proses pengurangan antara nilai makespan yang dihasilkan dengan suatu nilai tetap. Nilai tetap yang akan digunakan merupakan salah satu nilai makespan yang dihasilkan dari solusi yang akan dianggap sebagai nilai tengah dari makespan yang mungkin dihasilkan. Solusi yang akan dijadikan nilai tengah ini merupakan solusi yang menggunakan urutan nama / nomor pekerjaan sebagai urutan pengerjaan. Sebagai contoh, jika terdapat 3 buah pekerjaan, maka solusi yang dijadikan nilai tengah adalah solusi dengan urutan pengerjaan [1,2,3].

Banyaknya feromon yang ditambahkan tidak akan bernilai negatif. Jika feromon yang ditambahkan bernilai negatif, maka penambahan feromon dari solusi tersebut tidak akan dilakukan. Penambahan feromon akan dilakukan pada indeks-indeks matriks tertentu berdasarkan solusi yang diambil. Sebagai contoh, jika solusi yang diambil adalah [2,1,3], maka indeks-indeks matriks yang akan ditambahkan feromonnya adalah indeks matriks [2,1] dan [1,3]. Dengan ditambahkannya matriks pada indeks-indeks tersebut, diharapkan indeks matriks tersebut akan lebih sering diacu dalam proses pengambilan jalur. Dengan ditambahkannya feromon pada indeks-indeks tersebut juga diharapkan solusi dengan urutan pengerjaan [2,1,3] akan lebih sering diambil.

3.3 Proses Optimisasi Penjadwalan Flow Shop Dengan Menggunakan Algoritma Ant Colony

Pada saat melakukan proses optimisasi, algoritma ant colony akan melakukan proses pelatihan secara berulang-ulang. Pada setiap proses pelatihan, algoritma ant colony menyebarkan sejumlah semut yang akan mengerjakan kasus flow shop sesuai solusi yang dipilih masing-masing semut. Solusi-solusi tersebut dipilih secara acak berdasarkan nilai feromon yang disimpan. Solusi solusi terbaik yang telah dipilih oleh suatu semut akan disimpan dan dibandingkan dengan solusi terbaik dari proses pelatihan sebelumnya.

Penambahan dan evaporasi nilai feromon yang disimpan akan dilakukan setiap suatu fase pelatihan dari algoritma ant colony selesai dilakukan. Nilai feromon yang ditambahkan akan

disesuaikan dengan solusi dan hasil yang didapat oleh masing-masing semut. Semakin baik hasil yang diberikan suatu solusi, semakin banyak nilai feromon yang ditambahkan pada indeks feromon yang mengacu pada solusi tersebut. Proses pelatihan akan terus dilakukan hingga suatu kondisi berhenti dicapai. Kondisi berhenti tersebut merupakan penanda bahwa hasil solusi yang dipilih sudah dapat dianggap sebagai solusi yang optimal.

Algoritma ant colony merupakan algoritma optimisasi yang cukup terkenal. Keunggulan-keunggulan dari algoritma ini diantaranya adalah :

- Mampu diaplikasikan pada berbagai macam permasalahan.
- Mampu memberikan salah satu solusi yang dapat dianggap baik secara cepat.

Sedangkan kekurangan algoritma ini adalah :

- Solusi terbaik pasti ditemukan, akan tetapi tidak ada lama waktu yang pasti untuk ditemukannya solusi optimal tersebut.
- Analisa secara teori tidak mungkin dilakukan, karena menggunakan pengambilan solusi secara acak dan tidak tetap.

Untuk menangani kelemahan-kelemahan tersebut, algoritma ant colony perlu menjabarkan jumlah pelatihan yang akan dilakukan dan banyaknya semut yang akan disebar pada satu kali proses pelatihan. Hal-hal tersebut perlu ditentukan agar proses pelatihan mampu menemukan solusi yang optimal dalam waktu yang relatif cepat. Semakin banyak proses pelatihan dan jumlah semut yang disebar, semakin besar kemungkinan ditemukannya solusi yang optimal. Akan tetapi pada saat yang bersamaan akan meningkatkan waktu pelaksanaan proses optimisasi. Baik atau tidaknya suatu proses optimisasi ditentukan oleh tingkat keoptimalan solusi yang dihasilkan dan waktu yang diperlukan untuk proses optimisasi.

3.3.1 Jumlah Semut

Jumlah semut dalam proses optimisasi algoritma ant colony sangat berpengaruh pada tingkat keoptimalan dari solusi yang dihasilkan. Semakin banyak jumlah semut yang disebar, maka tingkat keoptimalan solusi yang diberikan akan cenderung lebih baik. Akan tetapi semakin banyak semut yang disebar akan menambah tingkat kompleksitas algoritma ini dan menambah waktu yang diperlukan untuk proses optimisasi. Jumlah semut yang disebar perlu ditentukan agar algoritma mampu memberikan solusi yang dapat dianggap cukup baik dalam waktu yang relatif cepat.

Pencarian solusi optimal untuk suatu kasus flow shop dapat dilakukan secara brute force. Pencarian solusi secara brute force dapat dianggap sebagai cara yang paling tidak efisien. Brute force akan menjabarkan solusi-solusi yang ada dan mencoba penyelesaian dengan menggunakan masing-masing solusi tersebut. Jika proses optimisasi diaplikasikan pada sebuah permasalahan, sebaiknya proses optimisasi tersebut dapat berjalan lebih cepat dibandingkan dengan cara brute force.

Jika permasalahan flow shop akan diselesaikan secara brute force, maka setiap solusi yang ada akan dijabarkan. Solusi-solusi pada flow shop dapat dicari dengan melakukan proses fungsi kombinasi pada banyaknya pekerjaan yang ada. Dilihat dari sisi kompleksitas, cara brute force memiliki tingkat kompleksitas n!, di mana n merupakan banyaknya pekerjaan yang ada.

Jumlah semut yang akan disebar sebaiknya kurang dari tingkat kompleksitas dengan cara brute force tersebut. Hal ini dilakukan agar optimisasi algoritma ant colony ini menjadi relatif lebih cepat dibandingkan dengan cara brute force. Jumlah semut yang optimal masih belum diketahui, oleh karena itu sekurang-kurangnya perlu diketahui pengaruh dari jumlah semut yang disebar terhadap hasil optimisasi algoritma ini.

3.3.2 Aturan Berhenti

Ada berbagai cara yang dapat dipakai dalam menentukan waktu berhenti dari proses optimisasi algoritma ant colony. Beberapa parameter yang dapat digunakan untuk penentuan waktu berhenti di antaranya

- Nilai data keluaran yang dihasilkan : algoritma akan terus melakukan pelatihan hingga data keluaran yang dihasilkan bernilai sama dengan data-data keluaran sebelumnya
- Sesuai dengan keinginan pengguna : algoritma akan terus melakukan pelatihan hingga pengguna merasa hasil akhir yang diberikan sudah cukup optimal.

Perlu diingat pula bahwa salah satu dari kekurangan algoritma ant colony adalah tidak adanya lama waktu yang pasti untuk ditemukannya solusi yang paling optimal. Algoritma ant colony menggunakan proses randomisasi untuk penentuan solusi-solusi optimal. Meskipun proses randomisasi tetap dipengaruhi oleh feromon-feromon pada indeks matriks, tidak berarti bahwa solusi yang lebih optimal akan selalu terpilih. Proses randomisasi pada algoritma tetap memiliki kemungkinan terpilihnya solusi yang kurang optimal secara terus menerus.

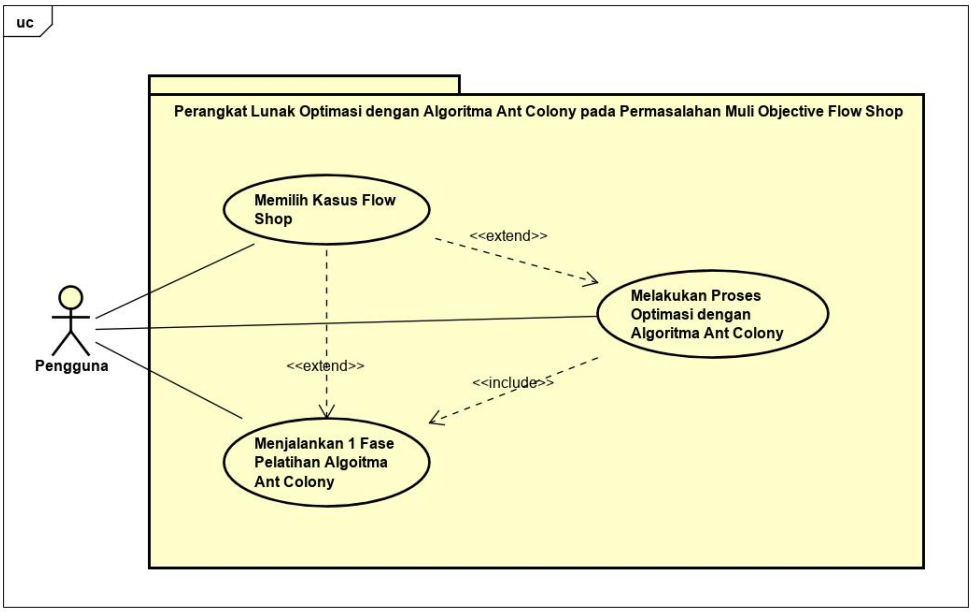
3.4 Analisis Perangkat Lunak

Dengan tujuan untuk mengatasi permasalahan yang dipaparkan dalam bab - bab sebelumnya, maka dirancang perangkat lunak yang mampu melakukan klasifikasi permasalahan flowshop tersebut. Perangkat lunak yang dibuat dapat melakukan fitur-fitur berikut ini :

- Perangkat lunak dapat membaca masukan data permasalahan flowshop yang berupa file .txt.
- Perangkat lunak dapat melakukan pelatihan menggunakan data dari Tailard.
- Perangkat lunak dapat menampilkan makespan dan urutan pengerjaan dari job yang ada.]item Perangkat lunak dapat menamilkan nilai utilitas dari mesin yang ada.

3.4.1 Use Case Diagram

Perangat Lunak yang dibuat akan mampu berinteraksi dengan pengguna. Pengguna akan mampu menjabarkan kasus flow shop yang ingin dicari hasil optimalnya. Pengguna juga akan dapat mampu mengatur kapan dan bagaimana berlangsungnya suatu proses optimisasi. Untuk mewujudkan interaksi tersebut, berikut adalah rancangan use case diagram yang menjabarkan hal-hal yang mampu dilakukan pengguna melalui perangkat lunak ini.



Gambar 3.1: Use Case Diagram

Penjelasan diagram :

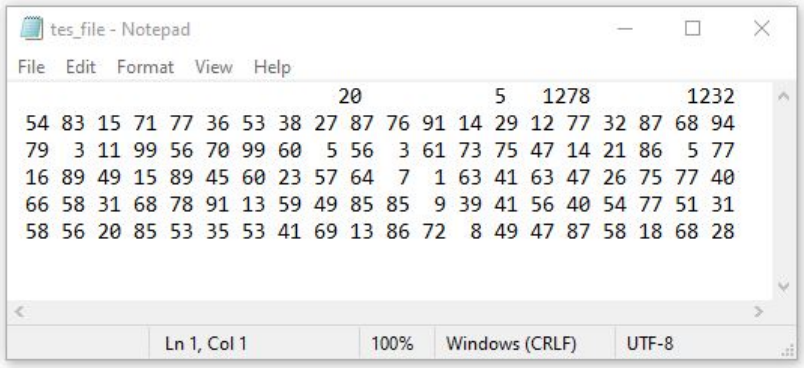
- Memilih kasus Flow Shop
  Pengguna dapat memilih kasus flow shop yang ingin dicari hasil optimalnya. Kasus tersebut akan dijabarkan dalam sebuah file teks dengan format penulisan tertentu. Pengguna dapat memilih salah satu file teks tersebut sebagai kasus flow shop yang ingin dicari hasil optimalnya. Pengguna hanya dapat melakukan proses optimisasi dengan menggunakan algoritma ant colony setelah pengguna memilih kasus flow shop yang ingin dioptimisasi.

- Menjalankan 1 fase pelatihan algoritma ant colony
  Pengguna dapat menjalankan 1 kali fase pelatihan dari algoritma ant colony. Proses pelatihan ini akan mengembalikan informasi mengenai hasil optimal (makespan dan nilai utilitas mesin) yang didapat pada fase pelatihan ini.

- Melakukan proses optimasi dengan menggunakan algoritma ant colony
  Proses optimisasi ini akan menjalankan fase pelatihan algoritma ant colony secara terus menerus hingga kondisi berhenti yang ditentukan tercapai. Hasil makespan dan nilai utilitas yang didapatkan pada setiap fase pelatihan akan ditampilkan.

3.4.2 Input dan Output

Perangkat lunak akan mampu menerima data masukan dari sebuah file teks. File teks tersebut akan berisi informasi mengenai banyaknya mesin, banyaknya pekerjaan, waktu proses untuk setiap pekerjaan, dan nilai upper beserta lower bound dari setiap kasus. File teks tersebut akan memiliki format penulisan tertentu, agar file teks tersebut dapat dikonversi menjadi suatu kasus flow shop.



Gambar 3.2: Input Program

Gambar di atas merupakan contoh file teks yang dapat diterima oleh program yang akan dibangun. File teks tersebut terdiri dari 20 pekerjaan dan 5 mesin. Penjelasan lengkap dari gambar tersebut sebagai berikut :

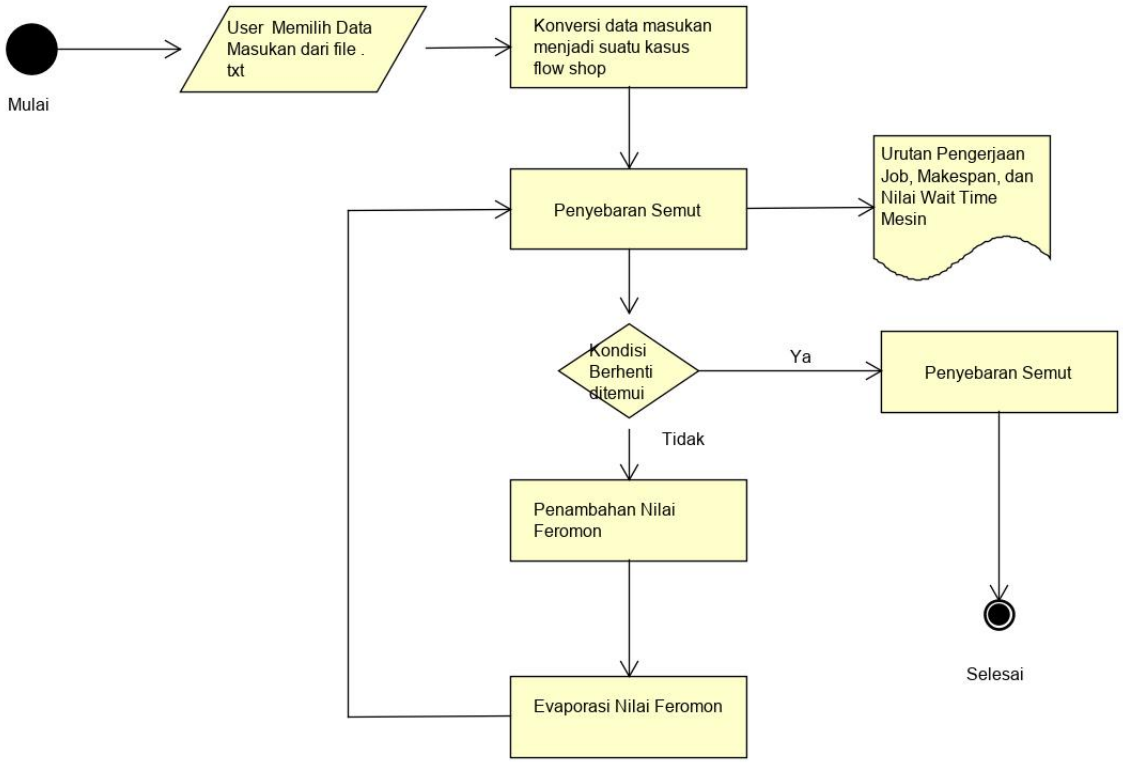
1. Baris pertama terdiri atas empat nilai.
2. Nilai pertama merupakan banyaknya pekerjaan.
3. Nilai kedua merupakan banyaknya mesin.
4. Nilai ketiga merupakan upper bound dari suatu kasus.
5. Nilai keempat merupakan lower bound dari suatu kasus.
6. Baris kedua hingga akhir merupakan waktu proses untuk setiap pekerjaan.

Nilai upper bound akan dianggap sebagai nilai makespan optimal / nilai makespan target yang ingin dicapai oleh proses optimisasi. Nilai lower bound akan dianggap sebagai nilai makespan minimal yang mungkin dicapai dari suatu kasus. Nilai lower bound ini akan dijadikan batas nilai makespan yang mungkin dihasilkan dari suatu kasus. Nilai lower bound ini akan digunakan untuk pengujian perangkat lunak. Jika nilai makespan yang dihasilkan lebih kecil dari nilai lower bound, maka kemungkinan terdapat kesalahan pada perangkat lunak.

Setelah mengkonversi data masukan menjadi suatu kasus flow shop, perangkat lunak akan melakukan proses optimisasi. Proses optimisasi akan menghasilkan suatu solusi berupa urutan pengerjaan yang menghasilkan makespan yang dianggap optimal beserta nilai wait time dari setiap kasus. Perangkat lunak kemudian akan menampilkan di layar pengguna mengenai urutan pengerjaan yang dianggap optimal, nilai makespan yang dihasilkan, dan nilai wait time dari suatu kasus. Perangkat lunak juga akan menampilkan banyaknya pelatihan yang telah dilakukan untuk mendapat urutan pengerjaan tersebut.

3.4.3 Flow Chart Proses Optimasi

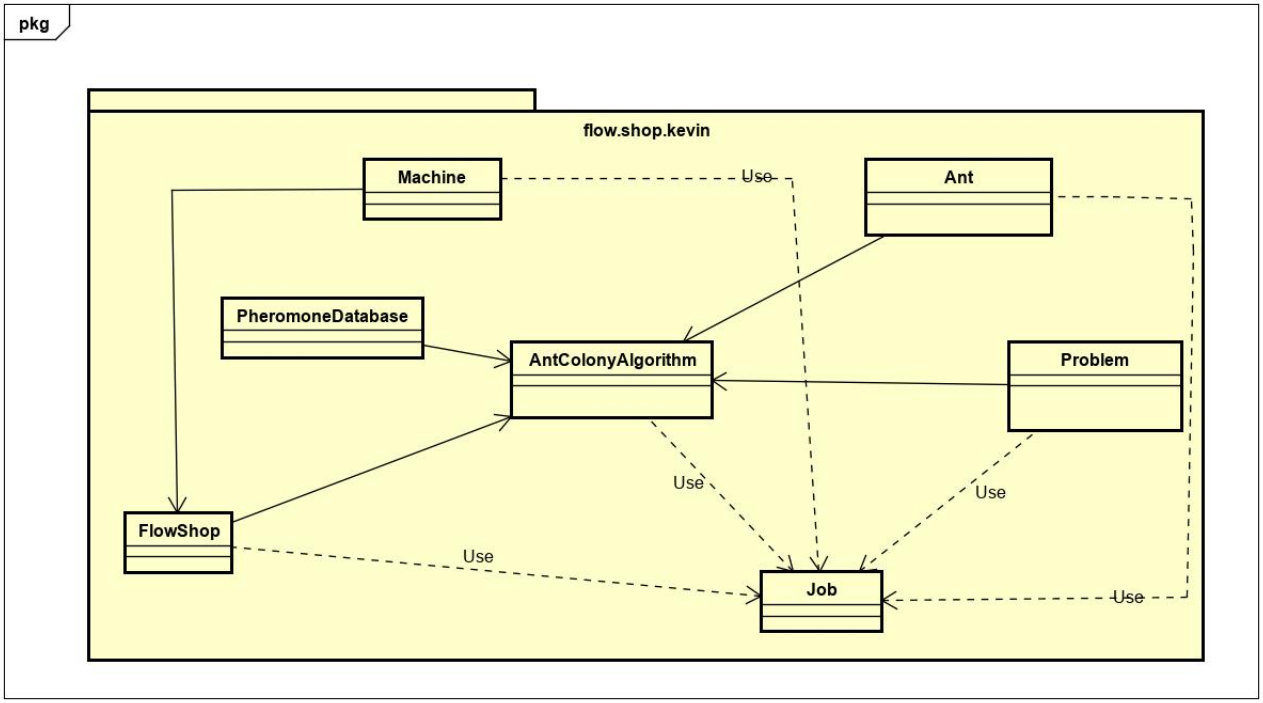
Proses optimisasi dengan algoritma ant colony dilakukan setelah mengkonversi data masukan menjadi sebuah kasus flow shop. Algoritma ant colony akan menyesuaikan diri dengan kasus flow shop yang ingin dicari solusi optimalnya. Algoritma ant colony akan melakukan proses optimisasi dengan cara melakukan proses pelatihan secara berulang-ulang hingga suatu kondisi berhenti tercapai. Berikut adalah flow chart mengenai bagaimana perangkat lunak ini akan melakukan proses optimisasi dengan menggunakan algoritma ant colony.



Gambar 3.3: Flowchart Optimasi

3.4.4 Diagram Kelas Secara Umum

Diagram kelas digunakan untuk memberikan gambaran mengenai kelas-kelas yang terdapat dalam sistem perangkat lunak yang dibangun serta hubungan antar kelasnya. Pada bab ini hanya akan ditampilkan diagram kelas secara umum saja. Untuk lebih detailnya akan dijelaskan pada bab selanjutnya. Diagram kelas tersebut sebagai berikut :



Gambar 3.4: Diagram kelas umum

BAB 4

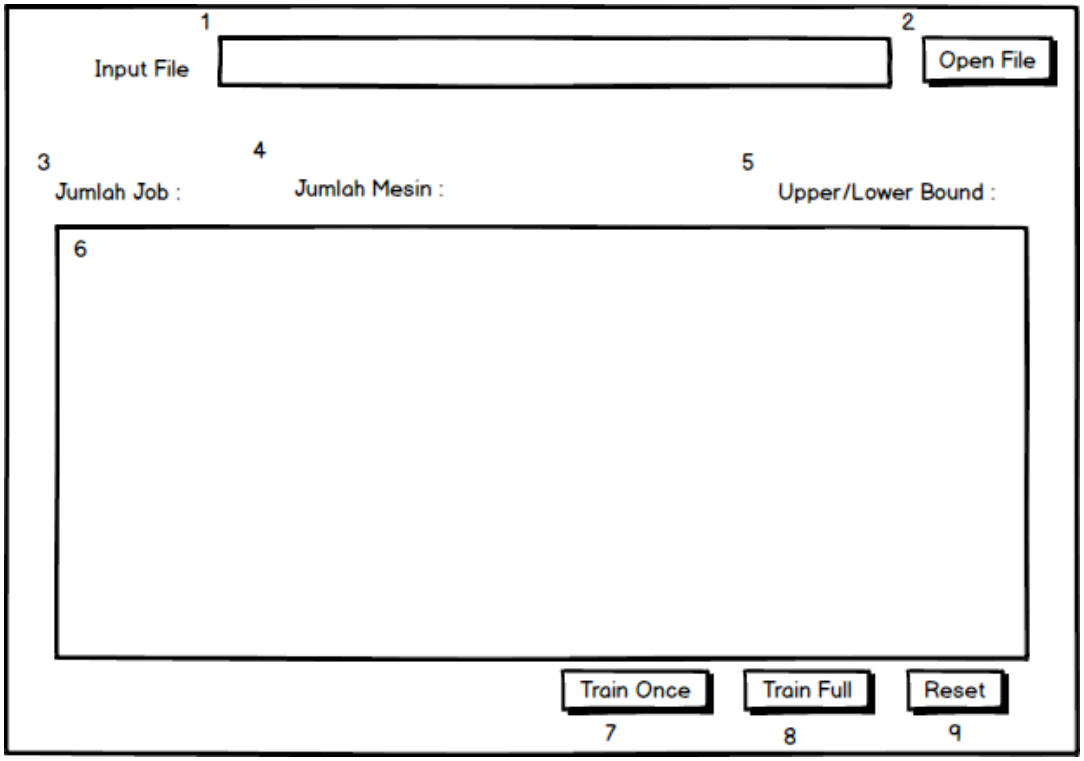
PERANCANGAN PERANGKAT LUNAK

Pada bab ini akan dibahas mengenai detil-detil dari perangkat lunak optimisasi algoritma ant colony untuk kasus multi objective flow shop. Struktur kelas dan hal-hal yang mampu dilakukan melalui perangkat lunak ini juga akan dijabarkan.

4.1 Perancangan Antarmuka Grafis

Untuk mempermudah penggunaan, perangkat lunak akan memiliki sebuah interface. Interface ini dibuat agar pengguna perangkat lunak dapat dengan mudah menjabarkan kasus flow shop yang ingin dioptimisasi. Interface ini juga dibuat agar proses optimisasi yang dilakukan dapat lebih mudah dipantau. Melalui interface ini, pengguna dapat memberikan data masukan kasus flow shop lewat sebuah file teks. Pengguna juga dapat mengetahui hasil optimisasi yang diberikan pada setiap fase pelatihan.

Berikut tampilan *interface* tersebut :



Gambar 4.1: Rancangan *interface* perangkat lunak

Keterangan dari *interface* tersebut sebagai berikut :

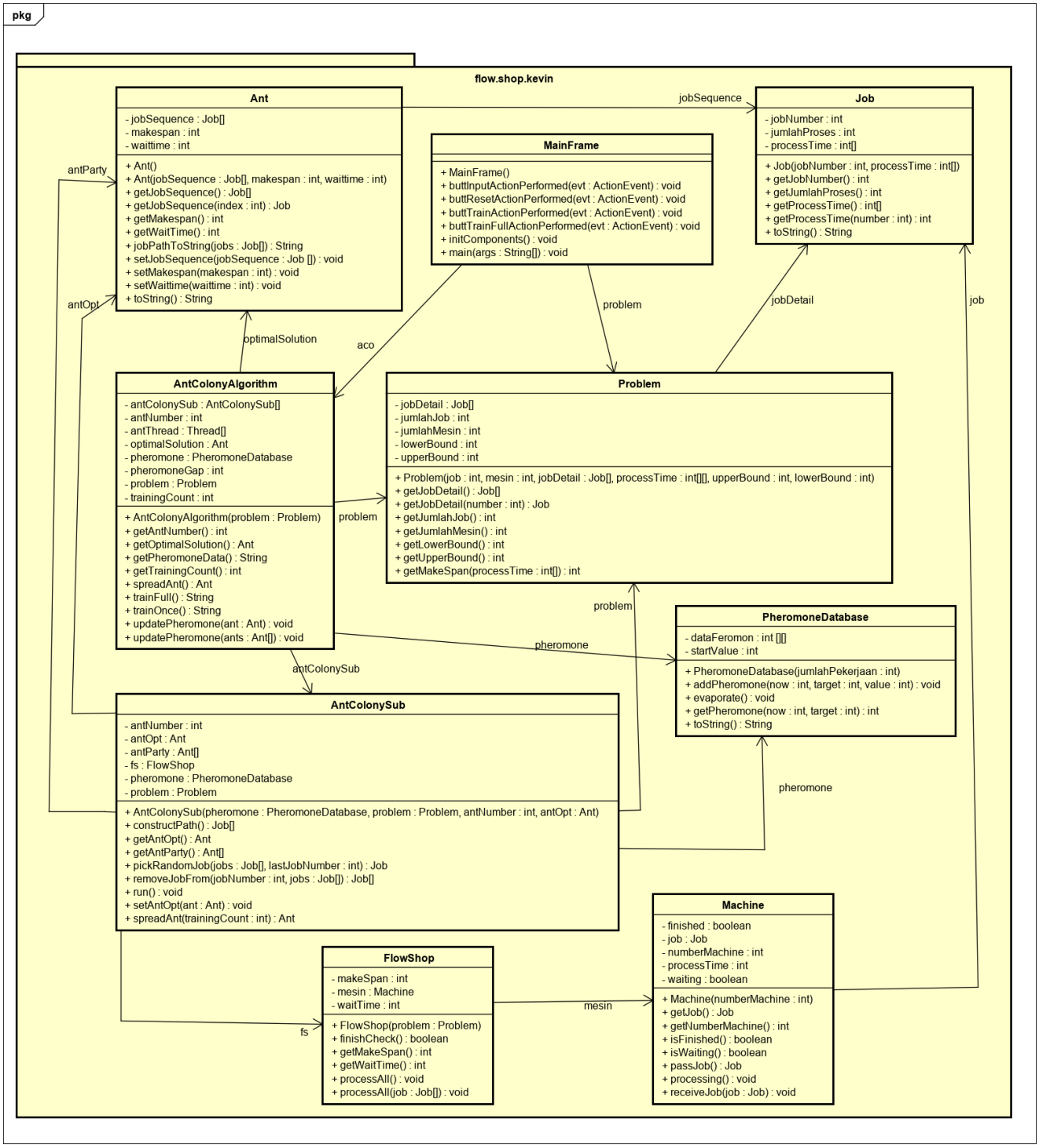
1. Memberitahu informasi nama *file* teks yang telah dipilih pengguna untuk dioptimisasi.

2. Memilih *file* teks yang berisi satu kasus flow shop. File yang dipilih harus berupa file.txt. Agar *file* dapat dikonversi menjadi sebuah kasus flow shop, data di dalam file tersebut juga harus sesuai dengan format penulisan yang sudah ditentukan.
3. Menampilkan banyaknya pekerjaan yang ada pada kasus flow shop yang dipilih.
4. Menampilkan banyaknya mesin yang ada pada kasus flow shop yang dipilih.
5. Menampilkan informasi nilai upper bound dan lower bound dari kasu flow shop yang dipilih.
6. Menampilkan informasi mengenai solusi optimal yang diberikan pada setiap proses pelatihan. Informasi yang diberikan berupa urutan pengerjaan yang dilakukan, nilai makespan yang dihasilkan,nilai waiting time mesin, dan pada proses pelatihan ke berapa solusi tersebut didapatkan.
7. Melakukan proses pelatihan dengan menggunakan algoritma ant colony sebanyak 1 kali pada kasus flow shop yang dipilih.
8. Melakukan proses pelatihan dengan menggunakan algoritma ant colony pada kasus flow shop yang dipilih hingga kondisi berhenti dicapai. Kondisi berhenti yang digunakan adalah diberikannya solusi optimal yang sama pada 10 proses pelatihan terakhir.
9. Mengulang kembali proses optimisasi dari fase pelatihan pertama.

4.2 Diagram Kelas Lengkap

Perangkat lunak akan dibuat secara *Object Oriented* . Kelas-kelas yang dibuat akan memperhatikan kebutuhan dan keterhubungan dari masing-masing objek yang terlibat. Interface yang akan dibuat akan langsung memiliki keterhubungan dengan kelas yang mampu melakukan proses optimisasi. Keterhubungan tersebut tidak diperantara oleh kelas Controller seperti pada topologi *Model* , *View* , dan *Controller* .

Berikut adalah diagram kelas dari perangkat lunak yang dibuat.



Gambar 4.2: Diagram kelas dari perangkat lunak

Penjelasan dari diagram kelas :

- **Kelas MainFrame**
  Kelas ini merupakan kelas yang membentuk *interface* dan mengaplikasikan fungsi-fungsi aplikasi pada komponen-komponen *interface* . *Interface* yang dibentuk oleh kelas ini berguna untuk proses interaksi dengan pengguna selama perangkat lunak dijalankan. Atribut dan method yang dimiliki oleh kelas ini adalah :

- Atribut
  - problem
    Kasus flow shop yang telah dikonversi dari file teks dan ingin dicari hasil optimalnya.
  - aco
    Algoritma ant colony yang digunakan untuk pencarian hasil optimal dari kasus flow shop yang disimpan pada atribut problem.
- Method
  - MainFrame
    Constructor dari kelas ini.
  - initComponents
    Menginisialisasikan komponen-komponen seperti Tombol, *Text Box* , dan *Text Area* pada *interface* .
  - buttInputActionPerformed
    Membuat kotak dialog baru untuk memilih file teks berisi kasus flow shop yang ingin dicari hasil optimalnya. Method ini akan mengkonversi file teks tersebut menjadi kasus flow dan mengkonfigurasi algoritma ant colony yang akan digunakan untuk pencarian hasil optimal dari kasus tersebut. Method ini akan dijalankan saat pengguna menekan tombol "Browse".
  - buttTrainActionPerformed
    Melakukan proses pelatihan dengan menggunakan algoritma ant colony sebanyak 1 kali pada kasus flow shop yang dipilih. Hasil dari proses optimisasi akan ditampilkan pada pengguna. Method ini akan dijalankan saat pengguna menekan tombol "Train Once".
  - buttResetActionPerformed
    Mengulang kembali proses optimisasi dari kondisi awal. Method ini akan dijalankan saat pengguna menekan tombol "Reset".
  - buttTrainFullActionPerformed
    Melakukan proses pelatihan dengan menggunakan algoritma ant colony pada kasus flow shop yang dipilih hingga kondisi berhenti dicapai. Kondisi berhenti yang digunakan adalah diberikannya solusi optimal yang sama pada 10 proses pelatihan terakhir. Hasil dari proses optimisasi hingga kondisi tercapainya kondisi berhenti akan ditampilkan pada pengguna. Method ini akan dijalankan saat pengguna menekan tombol "Train Full".
  - main
    Method untuk menginisialisasi dan menjalankan perangkat lunak.

- **Kelas Job**
  Kelas ini merepresentasikan sebuah pekerjaan yang akan dikerjakan pada sebuah kasus flow shop. Kelas ini menyimpan informasi-informasi mengenai sebuah pekerjaan pada kasus flow shop. Atribut dan method yang dimiliki oleh kelas ini adalah :

  - Atribut
    - jobNumber
      Menyimpan nomor petunjuk identitas sebuah pekerjaan. Atribut ini merupakan nomor pembeda sebuah pekerjaan dari pekerjaan lainnya dalam sebuah kasus flow shop.
    - processTime
      Menyimpan detil waktu pengerjaan proses-proses pada sebuah pekerjaan. Detil-detil waktu proses tersebut disimpan dalam bentuk array integer. Indeks array berfungsi sebagai penunjuk nomor proses dari pekerjaan tersebut.

    * jumlahProses
      Menyimpan detil mengenai banyaknya proses dalam sebuah pekerjaan. Nilai atribut ini diberikan pada saat pembentukkan kelas. Nilai dari atribut ini sesuai dengan panjang array integer dari atribut processTime.
  - Method
    * job
      Contructor dari kelas ini. Pembuatan kelas ini dilakukan saat mengkonversi file teks menjadi suatu kasus flow shop. Adapun parameter-parameter yang diperlukan untuk pembentukkan kelas ini adalah :
      · jobNumber : nomor / nama pekerjaan.
      · processTime : detil waktu proses pekerjaan, dalam bentuk array integer.
    * toString
      Method ini akan mengembalikan detil waktu-waktu proses pada sebuah pekerjaan. Detil waktu tersebut akan ditampilkan dalam sebuah String yang akan diberikan kepada pengguna perangkat lunak.
    * getJobNumber
      Method ini akan mengembalikan nilai atribut jobNumber yang berisi nomor identitas dari pekerjaan.
    * getProcessTime
      Method ini akan mengembalikan nilai atribut processTime yang berisi detil waktu proses dari pekerjaan ini. Atribut ini akan diberikan dalam bentuk array integer. Method ini juga mampu menerima parameter sebuah angka integer sebagai penunjuk nomor proses. Jika method ini menerima parameter nomor proses tersebut, maka method ini akan mengembalikan sebuah integer lama waktu pengerjaan proses ke-x dari pekerjaan tersebut, di mana x tersebut merupakan parameter nomor pekerjaan yang dimasukkan.
    * getJumlahProses
      Method ini akan mengembalikan nilai atribut jumlahProses yang berisi banyaknya proses pada pekerjaan ini.

- **Kelas Problem**
  Kelas ini merepresentasikan sebuah kasus flow shop. Kelas ini menyimpan informasi - informasi pada suatu kasus flow shop. Atribut dan method yang dimiliki oleh kelas ini adalah :

  - Artribut
    * jumlahJob
      Menyimpan informasi mengenai banyaknya pekerjaan pada sebuah kasus flow shop.
    * jumlahMesin
      Menyimpan informasi mengenai banyaknya perangkat mesin yang mampu mengerjakan masing-masing proses.
    * jobDetail
      Menyimpan informasi mengenai setiap pekerjaan yang terdapat pada kasus flow shop. Informasi disimpan dalam bentuk array kelas Job. Indeks array berfungsi sebagai penunjuk nomor pekerjaan.
    * upperBound
      Menyimpan informasi nilai upper bound dari kasus flow shop.
    * lowerBound
      Menyimpan informasi nilai lower bound dari kasus flow shop.
  - Method

* Problem
  Constructor dari kelas ini. Pembuatan kelas ini merupakan hasil konversi file teks menjadi sebuah kasus flow shop. Parameter-parameter yang dibutuhkan untuk pembentukkan kelas ini adalah :
  - job : banyaknya pekerjaan.
  - mesin : banyaknya perangkat mesin.
  - jobDetail : informasi pekerjaan-pekerjaan yang ada, dalam bentuk array kelas Job.
  - processTime : pembentukan waktu proses dari setiap job, berdasarkan mesin dan job. Akan digunakan pada kelas MainFrame.
  - upperBound : nilai upper bound.
  - lowerBound : nilai lower bound.
* getJumlahJob
  Mengembalikan nilai atribut jumlahJob yang berisi informasi mengenai banyaknya pekerjaan pada kasus flow shop.
* getJobDetail
  Mengembalikan detil informasi mengenai pekerjaan yang ada pada kasus flow shop. Method ini akan mengembalikan informasi detil pekerjaan dalam bentuk array kelas Job. Method ini juga mampu menerima sebuah parameter number, di mana number merupakan penunjuk nomor / nama pekerjaan. Jika method ini menerima parameter tersebut, maka method ini akan mengembalikan informasi pekerjaan dengan nomor / nama pekerjaan tersebut. Informasi tersebut diberikan dalam bentuk sebuah objek dari kelas Job.
* getJumlahMesin
  Mengembalikan detil informasi mengenai banyaknya perangkat mesin yang ada pada kasus flow shop.
* getUpperBound
  Mengembalikan nilai upper bound pada kasus flow shop yang disimpan pada atribut upperBound
* getLowerBound
  Mengembalikan nilai lower bound pada kasus flow shop yang disimpan pada atribut lower Bound.

- **Kelas Machine**
  Kelas ini merepresentasikan sebuah mesin pada sistem penjadwalan flow shop. Kelas ini menjabarkan kondisi mesin yang sedang mengerjakan / menunggu sebuah pekerjaan. Atribut dan method yang dimiliki oleh kelas ini adalah :

  – Atribut
  * job
    Menyimpan informasi mengenai pekerjaan yang sedang dikerjakan / disimpan pada mesin ini. Informasi disimpan dalam bentuk sebuah objek dari kelas Job.
  * numberMachine
    Menyimpan informasi mengenai banyaknya mesin.
  * processTime
    Menyimpan informasi mengenai lama waktu proses yang telah dilalui dalam pengerjaan proses dari sebuah pekerjaan. Jika pekerjaan telah selesai dikerjakan / tidak ada pekerjaan pada mesin, maka atribut ini akan bernilai 0.
  * finished Menyimoan informasi mengenai status mesin. Atribut ini mengindentifikasikan sebuah mesin telah menyelesaikan pekerjaannya.

  * waiting
    Menyimpan informasi mengenai status mesin. Atribut ini mengindentifikasikan sebuah mesin sedang kosong dan menunggu pekerjaan baru untuk dikerjakan.

- Method
  * Machine
    Constructor dari kelas ini. Pembuatan kelas ini dilakukan pada saat konfigurasi mesin-mesin pada sistem flow shop agar sesuai dengan kasus flow shop yang ingin dikerjakan / dioptimisasi. Parameter yang diperlukan dalam pembentukkan kelas ini adalah :
    · numberMachine : banyaknya mesin yang ada pada kasus flow shop
    Pada saat pembentukkan objek dari kelas ini, mesin akan dianggap sebagai sebuah mesin baru tanpa ada pekerjaan apapun yang dikerjakan di dalamnya. Oleh karena itu, nilai dari atribut job akan bernilai null. Nilai dari atribut processTime akan bernilai 0. Nilai dari atribut *waiting* dan *finished* akan bernilai *true*.
  * processing
    Menambah 1 nilai pada atribut processTime. Jika nilai dari atribut processTime telah sama dengan waktu pengerjaan dari proses pekerjaan yang ditugaskan pada mesin, maka method ini akan mengubah status mesin menjadi sudah selesai mengerjakan tugasnya. Status mesin yang sudah selesai mengerjakan tugasnya ditandai dengan atribut *finished* yang bernilai *true*. Jika mesin telah menyelesaikan tugasnya, maka nilai dari atribut processTime juga akan diatur ulang menjadi bernilai 0. Method ini hanya akan berjalan pada mesin dengan nilai atribut finished *false*.
  * passJob
    Mengoper pekerjaan yang telah diselesaikan oleh mesin. Method ini akan mengembalikan pekerjaan yang telah diselesaikan oleh mesin dalam bentuk sebuah objek dari kelas Job. Pada saat yang sama, method ini juga akan mengubah status mesin menjadi kosong / menunggu pekerjaan baru. Mesin yang sedang menunggu pekerjaan baru ditandai dengan berubahnya nilai dari atribut *waiting* menjadi *true*. Method ini juga akan mengatur ulang nilai dari atribut job menjadi null. Method ini hanya bisa dijalankan pada mesin dengan nilai atribut *finished true* dan nilai atribut *waiting false*. Nilai atribut tersebut dapat diartikan sebagai mesin yang sudah menyelesaikan pekerjaannya tetapi masih menyimpan pekerjaannya.
  * receiveJob
    Menerima pekerjaan baru untuk dikerjakan mesin. Method ini menerima sebuah parameter berupa objek dari kelas Job, di mana parameter tersebut merupakan pekerjaan baru yang harus dikerjakan oleh mesin. Parameter tersebut akan disimpan sebagai atribut job dari mesin. Method ini akan mengubah status mesin menjadi sedang mengerjakan sebuah pekerjaan. Mesin yang sedang mengerjakan sebuah pekerjaan ditandai dengan atribut *finished* dan *waiting* yang bernilai false. Method ini hanya dapat dijalankan pada mesin yang sedang menunggu pekerjaan baru(atribut waiting bernilai *true*).
  * isWaiting
    Mengembalikan informasi mengenai apakah mesin sedang kosong atau tidak. Informasi didapat dari atribut waiting. Jika mesin sedang kosong dan sedang menunggu diberikannya pekerjaan baru, maka method ini akan mengembalikan nilai boolean true. Jika mesin sedang mengerjakan / menyimpan sebuah pekerjaan, maka method akan mengembalikan nilai boolean false.
  * isFinished
    Mengembalikan informasi mengenai apakah mesin sudah / belum menyelesaikan pekerjaannya. Informasi tersebut didapatkan dari atribut finished. Jika mesin

telah menyelesaikan proses dari pekerjaan di dalamnya, maka method ini akan mengembalikan nilai boolean true. Jika mesin belum menyelesaikan pekerjaannya, maka method akan mengembalikan nilai boolean false.

* getJob
  Mengembalikan informasi mengenai pekerjaan yang sedang dikerjakan / disimpan oleh mesin. Informasi diberikan dalam bentuk sebuah objek dari kelas Job. Jika mesin sedang tidak mengerjakan / menyimpan sebuah pekerjaan, maka method ini akan mengembalikan nilai null.
* getNumberMachine
  Mengembalikan informasi mengenai banyaknya mesin pada kasus flow shop.

- **Kelas Flow Shop**
  Kelas ini merepresentasikan sebuah sistem penjadwalan flow shop. Kelas ini bertugas mengatur kapan sebuah mesin harus menerima dan mengoper sebuah pekerjaan. Kelas ini juga bertugas untuk menghitung lama waktu yang dibutuhkan untuk pengerjaan sekumpulan pekerjaan dengan urutan tertentu. Atribut dan method yang dimiliki oleh kelas ini adalah :

  – Atribut

  * mesin
    Informasi mengenai rangkaian mesin yang ada pada sistem flow shop. Informasi disimpan dalam bentuk array 1 dimensi dari kelas Mesin. Indeks dari array menunjukkan nomor mesin. Sebagai contoh indeks array [2] dari atribut ini menunjuk pada mesin kedua .
  * makespan
    Menyimpan informasi mengenai nilai makespan yang dihasilkan dari urutan pengerjaan.
  * waittime
    Menyimpan informasi mengenai nilai waktu tunggu mesin dalam memproses suatu kasus flow shop.

  – Method

  * FlowShop
    Constructor dari kelas ini. Pembentukkan kelas ini dilakukan pada saat konfigurasi sistem flow shop agar sesuai dengan kasus flow shop yang ingin dikerjakan / dioptimisasi. Parameter yang dibutuhkan untuk pembentukkan kelas ini adalah :
    · problem : kasus flow shop yang ingin dikerjakan / dioptimisasi.
    Pada saat pembentukkan sebuah sistem flow shop, konfigurasi rangkaian mesin pada sistem flow shop akan dilakukan. Banyaknya perangkat mesin untuk setiap proses akan diatur berdasarkan nilai-nilai yang ada pada parameter kasus flow shop. Nomor proses yang harus dikerjakan masing-masing mesin juga akan diatur secara otomatis.
  * processAll
    Method ini akan menghitung nilai makespan dan nilai waittime yang dihasilkan oleh parameter ar- ray kelas Job yang dimasukkan. Parameter tersebut dianggap sebagai sekumpulan pekerjaan yang akan dimasukkan / dikerjakan oleh sistem flow shop ini. Pekerjaan-pekerjaan tersebut akan dikerjakan secara terurut berdasarkan indeks ar- ray kelas Job tersebut. Method ini akan menghitung nilai makespan dan nilai waittime yang dihasilkan dari suatu solusi / urutan pengerjaan berdasarkan berapa kali method processAll dijalankan.
    Method ini akan mengatur kapan proses pengoperan dan penerimaan pekerjaan antar mesin terjadi. Method ini juga akan mengatur mesin mana yang berhak melakukan pengoperan / menerima sebuah pekerjaan. Proses pengoperan dimulai dengan

pengecekan mesin-mesin yang mengerjakan proses terakhir dari suatu pekerjaan. Jika mesin-mesin tersebut telah menyelesaikan pekerjaannya, maka mesin tersebut akan langsung mengoper pekerjaannya tanpa harus ada mesin lain yang menerima pekerjaan yang dioper. Pekerjaan yang telah dioper oleh mesin terakhir ini dianggap telah selesai dikerjakan. Jika mesin - mesin belum selesai mengerjakan pekerjaannya maka atribut waittime akan bertambah terus .
Jika terdapat mesin untuk proses selanjutnya yang sedang menunggu pekerjaan baru, maka pekerjaan dapat langsung dioper pada mesin tersebut. Jika tidak ada mesin untuk proses selanjutnya yang sedang menunggu, maka proses pengoperan belum akan dilakukan. Prioritas mengenai mesin yang berhak melakukan pengoperan terlebih dahulu ditentukan berdasarkan nomor / nama dari pekerjaan yang akan dioper oleh mesin. Jika suatu pekerjaan dikerjakan terlebih dahulu, maka pekerjaan tersebut akan memiliki prioritas lebih besar untuk dioper terlebih dahulu.
Method untuk menghitung nilai makespan dan waittime ini akan terus berjalan selama masih ada objek dari parameter array kelas Job yang belum dimasukkan ke dalam sistem flow shop. Method ini juga akan terus berjalan selama method finishCheck mengembalikan nilai true.

* finishCheck
Method ini berguna sebagai penanda berhentinya method processAll. Method ini akan memeriksa kondisi setiap mesin pada sistem flow shop. Jika setiap mesin sedang dalam kondisi menunggu pekerjaan baru (atribut waiting dari kelas Mesin bernilai true), maka seluruh pekerjaan yang diberikan dari method processAll dapat dianggap telah selesai dikerjakan. Jika seluruh pekerjaan telah selesai dikerjakan, maka method ini akan mengembalikan nilai true. Sebaliknya jika terdapat salah satu mesin yang tidak dalam kondisi menunggu pekerjaan baru, maka method ini akan mengembalikan nilai false.
* getMakeSpan
Mengembalikan informasi nilai makespan yang dihasilkan oleh urutan pengerjaan / solusi.
* getWaitTime
Mengembalikan informasi nilai waiitime yang dihasilkan oleh urutan pengerjaan / solusi.

- **Kelas Ant**
Kelas ini merepresentasikan seekor semut pada algoritma ant colony. Kelas ini akan menyimpan sebuah solusi dari hasil proses pembentukan solusi milik algoritma ant colony. Nilai-nilai yang disimpan dari seekor semut akan disesuaikan dengan permasalahan yang ingin diselesaikan. Dalam kasus ini, nilai yang akan disimpan oleh semut sebagai solusi akan disesuaikan dengan permasalahan flow shop. Atribut dan method dari kelas ini adalah :

  - Atribut
    * jobSequence
    Menyimpan informasi mengenai salah satu solusi / urutan pengerjaan dari kasus flow shop. Informasi tersebut disimpan dalam bentuk array kelas Job. Indeks array menunjukkan nomor urutan pengerjaan.
    * makespan
    Menyimpan informasi mengenai nilai makespan yang dihasilkan dari urutan pengerjaan / solusi yang disimpan pada atribut jobSequence.
    * waittime
    Menyimpan informasi mengenai nilai waktu tunggu mesin dalam memproses suatu kasus flow shop.

- Method
  - Ant
    Constructor dari kelas ini. Pembentukkan kelas ini dilakukan setiap dijalankannya fase pelatihan dari algoritma ant colony. Objek dari kelas ini mampu dibentuk langsung dengan parameter urutan pengerjaan dan nilai makespan yang ingin disimpan. Objek dari kelas ini juga mampu dibentuk tanpa dengan memberikan parameter apapun. Jika objek dibentuk tanpa dengan memasukkan parameter, maka atribut jobSequence akan bernilai null dan atribut makespan akan bernilai -1.
  - toString
    Mengembalikan informasi mengenai solusi yang disimpan oleh semut ini secara lengkap. Informasi akan diberikan dalam bentuk 3 baris string. Baris pertama menunjukkan nilai makespan yang dimiliki, baris kedua merupakan nilai dari waittime dan baris ketiga menunjukkan solusi / urutan pengerjaan yang menghasilkan nilai makespan tersebut.
  - jobPathToString
    Mengkonversi array kelas Job menjadi sebuah string urutan pengerjaan. Pengerjaan sebuah pekerjaan ditandai oleh atribut jobNumber dari kelas Job. Perlu diketahui pula bahwa perangkat lunak ini membentuk atribut jobNumber pada kelas Job secara otomatis dimulai dari nilai 0. Untuk mempermudah penggunaan perangkat lunak, method ini akan mengkonversi nilai jobNumber dalam string yang akan diberikan. Untuk setiap jobNumber pada string tersebut akan ditambahkan 1. Hal ini dilakukan agar pengerjaan pekerjaan pertama didefinisikan dengan jobNumber 1 bukan dengan jobNumber 0.
  - getMakespan
    Mengembalikan informasi nilai makespan yang dihasilkan oleh urutan pengerjaan / solusi pada atribut jobSequence.
  - getJobSequence
    Mengembalikan informasi urutan pengerjaan yang disimpan oleh semut. Informasi diberikan dalam bentuk array kelas Job, dengan indeks array sebagai nomor urutan pengerjaannya. Method ini juga mampu menerima sebuah parameter index. Jika method dijalankan dengan memberikan parameter tersebut, maka method akan mengembalikan informasi mengenai pekerjaan yang dikerjakan di urutan ke index. Informasi tersebut diberikan dalam bentuk seuah objek dari kelas Job.
  - getWaittime
    Mengembalikan informasi nilai waiitime yang dihasilkan oleh urutan pengerjaan / solusi pada atribut jobSequence.
  - setJobSequence
    Mengubah nilai dari atribut jobSequence sesuai dengan parameter array kelas Job yang dimasukkan.
  - setMakespan
    Mengubah nilai dari atribut makespan sesuai dengan nilai parameter yang dimasukkan.
  - setWaittime
    Mengubah nilai dari atribut waittime sesuai dengan nilai parameter yang dimasukkan.

- **Kelas PheromoneDatabase**
  Kelas ini merepresentasikan tempat penyimpanan nilai feromon untuk panduan proses pembentukan solusi secara acak milik algoritma ant colony. Cara penyimpanan nilai feromon disesuaikan dengan permasalahan yang ingin diselesaikan. Dalam kasus ini, nilai feromon yang disimpan akan disesuaikan dengan permasalahan flow shop dan tata cara yang digunakan

dalam proses pembentukan solusi secara acak milik algoritma ant colony. Atribut dan method yang dimiliki oleh kelas ini adalah :

- Atribut
  - dataFeromon
    Menyimpan informasi mengenai nilai-nilai feromon yang disimpan. Informasi tersebut disesuaikan dengan tata cara pembentukkan solusi acak dari algoritma ant colony. Informasi disimpan dalam bentuk array 2 dimensi. Indeks pertama dari array menunjukkan nomor pekerjaan sebelumnya dan indeks kedua menunjukkan nomor pekerjaan selanjuntya.
  - startValue
    Menyimpan informasi mengenai nilai awal yang diberikan pada masing-masing indeks atribut dataFeromon. Nilai awal tersebut hanya akan diberikan pada saat pembentukkan objek dari kelas ini.
- Method
  - PheromoneDatabase
    Constructor dari kelas ini. Pembentukkan kelas ini dilakukan saat pembentukkan kelas AntColonyAlgorithm untuk disesuaikan dengan kasus flow shop yang ingin dicari hasil optimalnya. Parameter yang diperlukan untuk pembentukkan kelas ini adalah :
    - jumlahPekerjaan : banyaknya pekerjaan pada kasus flow shop yang ingin dioptimisasi.

    Atribut dataFeromon dari kelas ini akan dibentuk berdasarkan nilai parameter jumlahPekerjaan yang dimasukkan. Array yang disimpan pada atribut dataFeromon akan berukuran n x n, di mana n merupakan nilai parameter jumlahPekerjaan yang dimasukkan. Masing-masing indeks dari atribut dataFeromon akan diberikan nilai awal sesuai dengan nilai dari atribut startValue.
  - getPheromone
    Mengembalikan nilai pada suatu indeks dari atribut dataFeromon. Method ini menerima 2 parameter: *now* dan *target*. Method ini akan mengembalikan nilai dari indeks array [now][target] dari atribut dataFeromon.
  - addPheromone
    Menambahkan nilai pada suatu indeks dari atribut dataFeromon. Method ini menerima 3 parameter: *now*, *target*, dan *value*. Method ini akan menambahkan nilai pada indeks array [now][target] dari atribut dataFeromon sejumlah nilai value.
  - evaporate
    Melakukan proses evaporasi feromon pada nilai-nilai feromon yang disimpan. Method ini akan mengurangi 30Nilai hasil evaporasi tidak akan lebih rendah dari 1.
  - toString
    Mengembalikan informasi mengenai nilai-nilai feromon yang disimpan pada atribut dataFeromon dalam bentuk string matirks 2 dimensi.

- **Kelas AntColonyAlgorithm**
  Kelas ini akan melakukan optimisasi dengan menggunakan algoritma ant colony. Kelas ini akan mencari urutan pengerjaan yang menghasilkan makespan yang optimal dalam suatu kasus flow shop. Dalam mencari hasil optimal tersebut, kelas ini akan dibantu oleh kelas PheromoneDatabase. Atribut dan method yang dimiliki oleh kelas ini adalah :

  - Atribut
    - problem
      Menyimpan kasus flow shop yang ingin dicari solusi optimalnya.

* pheromone
  Menyimpan kelas pheromoneDatabase yang akan membantu dalam pembentukkan solusi acak algoritma ant colony.
* optimalSolution
  Menyimpan solusi optimal dalam bentuk objek dari kelas Ant. Solusi optimal yang disimpan merupakan solusi terbaik dari fase-fase pelatihan yang telah dilakukan.
* trainingCount
  Menyimpan informasi mengenai banyaknya fase pelatihan yang telah dijalankan.
* pheromoneGap
  Menyimpan informasi mengenai sebuah nilai makespan yang dianggap sebagai nilai tengah dari nilai-nilai makespan lainnya yang mungkin dihasilkan. Nilai ini berguna dalam menentukan jumlah feromon yang akan ditambahkan dari sebuah solusi acak yang telah dibentuk.
* antColonySub
  Menyimpan informasi mengenai kelompok-kelompok semut yang ada dalam sistem algoritma ant colony ini. Masing-masing kelompok akan dijalankan secara bersamaan pada saat proses pencarian solusi-solusi lokal dari algoritma ant colony.
* antNumber
  Menyimpan informasi mengenai banyaknya semut pada setiap kelompok semut.
* antThread
  Menyimpan informasi thread yang dapat dijadikan penanda selesainya proses pencarian solusi / pelatihan dari salah satu kelompok semut.

– Method

* AntColonyAlgorithm
  Constructor dari kelas ini. Pembentukkan kelas ini dilakukan setelah selesai mengkonversi file teks menjadi suatu kasus flow shop. Parameter untuk pembentukkan kelas ini adalah :
  · problem : kasus flow shop yang ingin dicari hasil optimalnya.
  Pembentukkan kelas ini akan mengkonfigurasi atribut pheromone dan fs agar sesuai dengan kasus flow shop yang ingin dicari hasil optimalnya. Nilai dari atribut antNumber akan dibentuk berdasarkan atribut jumlahPekerjaan pada parameter kelas Problem yang dimasukkan. Nilai dari atribut pheromoneGap akan disamakan dengan nilai makespan yang dihasilkan dari salah satu solusi kasus flow shop. Solusi / urutan pengerjaan yang akan dipakai nilai makespan-nya adalah solusi yang urutan pengerjaannya terurut berdasarkan nilai atribut jobNumber.
* trainOnce
  Method ini akan menjalankan proses pelatihan pada algoritma ant colony sebanyak satu kali. Method ini akan menjalankan method spreadAnt dengan atribut antNumber sebagai parameter. Method ini akan mengembalikan hasil dari proses optimisasi tersebut dalam bentuk 3 baris string. Baris pertama menginformasikan banyaknya fase pelatihan yang telah dilakukan, baris kedua menunjukkan nilai makespan optimal yang diketahui hingga proses pelatihan ini beserta nilai waittime, dan baris ketiga menunjukkan urutan pengerjaan yang menghasilkan nilai makespan tersebut.
* trainFull
  Method ini akan menjalankan proses pelatihan pada algoritma ant colony hingga suatu kondisi berhenti tercapai. Method ini akan menjalankan method spreadAnt hingga suatu kondisi berhenti tercapai. Method spreadAnt dijalankan dengan atribut antNumber sebagai parameter. Kondisi berhenti yang digunakan adalah diberikannya hasil optimisasi yang sama pada 10 fase pelatihan terakhir. Method ini akan mengembalikan hasil dari proses optimisasi tersebut dalam bentuk sebuah

string. String tersebut akan berisi informasi mengenai hasil optimisasi yang diberikan pada setiap fase pelatihan hingga kondisi berhenti tersebut tercapai. Masing-masing hasil optimisasi pada tiap fase pelatihan dijabarkan dalam 3 baris string. Baris pertama menginformasikan nomor fase pelatihan, baris kedua menunjukkan nilai makespan optimal yang diketahui hingga proses pelatihan tersebut beserta nilai waittime, dan baris ketiga menunjukkan urutan pengerjaan yang menghasilkan nilai makespan tersebut.

* spreadAnt
  Method ini akan melakukan fase pelatihan pada algoritma ant colony. Method ini akan menginstruksikan masing-masing objek kelas AntColonySub pada atribut untuk mulai mencari solusi-solusi lokal. Secara terurut hal-hal yang dilakukan oleh method ini adalah :
  - Menginstruksikan setiap kelompok semut (atribut antColonySub) untuk memulai mencari solusi-solusi dari permasalahan pada atribut problem.
  - Masing-masing kelompok semut akan :
    - Membentuk solusi-solusi - Mencari nilai makespan dari solusi-solusi tersebut - Menyimpan solusi terbaik dan solusi-solusi lainnya yang dibentuk.
  - Menunggu setiap kelompok semut untuk menyelesaikan proses pencarian solusinya.
  - Melakukan update nilai feromon berdasarkan solusi-solusi yang dibentuk pada setiap kelompok semut.
  - Membandingkan solusi terbaik saat ini dengan solusi terbaik pada setiap kelompok semut.
  - Jika terdapat solusi yang lebih baik, maka solusi tersebut akan disimpan.
  - Menginformasikan solusi terbaik pada fase pelatihan ini pada setiap kelompok semut.
  - Melakukan evaporasi nilai feromon.
  - Menambah jumlah hitungan fase pelatihan.
  - Memberikan solusi optimal pada fase pelatihan ini sebagai data keluaran.
* updatePheromone
  Method ini akan melakukan proses penambahan nilai feromon berdasarkan solusi yang diberikan sebagai parameter. Method ini menerima parameter dari kelas Ant yang dapat dianggap sebagai suatu solusi untuk kasus flow shop. Indeks - indeks matriks feromon yang akan ditambahkan ditentukan berdasarkan atribut jobSequence milik parameter objek Ant.
  Jumlah nilai feromon yang akan ditambahkan pada setiap target indeks matriks ditentukan berdasarkan nilai makespan yang dihasilkan oleh solusi tersebut. Nilai feromon yang akan ditambahkan sejumlah selisih dari nilai makespan dari solusi ini dengan makespan dari solusi yang dianggap sebagai nilai tengah. Nilai makespan yang dianggap sebagai nilai tengah disimpan pada atribut pheromoneGap. Method ini juga mampu menerima array kelas Ant sebagai parameter. Jika method ini menerima array kelas Ant sebagai parameter, maka method ini akan melakukan penambahan nilai feromon untuk setiap solusi yang dimiliki oleh setiap objek Ant pada array tersebut.
* getTrainingCount
  Mengembalikan nilai dari atribut trainingCount yang merupakan banyaknya proses pelatihan yang telah dilakukan.
* getOptimalSolution
  Mengembalikan nilai dari atribut optimalSolution yang merupakan informasi solusi

optimal dari kasus flow shop pada atribut problem. Solusi optimal tersebut merupakan solusi terbaik dari fase-fase pelatihan yang telah dilakukan. Informasi solusi optimal tersebut diberikan dalam bentuk sebuah objek dari kelas Ant.

* getAntNumber
  Mengembalikan nilai dari atribut antNumber yang merupakan informasi mengenai banyaknya semut yang dibentuk pada setiap fase pelatihan.
* getPheromoneData
  Mengembalikan informasi dari nilai-nilai feromon yang digunakan untuk pembentukkan solusi acak dari algoritma ant colony. Informasi tersebut diolah dan dikembalikan dalam bentuk sebuah string matriks 2 dimensi.

- **Kelas AntColonySub**
  Kelas ini merepresentasikan sebuah kelompok semut pada algoritma ant colony. Kelas ini mengimplementasikan kelas *interface Runnable* agar mampu dijalankan secara bersamaan. Kelas ini akan dijalankan pada saat proses pelatihan algoritma ant colony dilakukan. Kelas ini akan mencari solusi-solusi lokal berdasarkan nilai-nilai feromon yang ada pada algoritma ant colony dan mencari nilai makespan yang dihasilkan berdasarkan solusi tersebut. Atribut dan method yang dimiliki oleh kelas ini adalah :

  – Atribut
    * problem
      Menyimpan kasus flow shop yang ingin dicari solusi optimalnya.
    * pheromone
      Menyimpan kelas pheromoneDatabase yang dioper dari kelas AntColonyAlgorithm yang akan membantu dalam pembentukkan solusi acak algoritma ant colony.
    * fs
      Sistem flow shop yang dibentuk berdasarkan atribut problem dan digunakan untuk menghitung nilai makespan dan nilai waittime yang dihasilkan oleh suatu solusi.
    * antOpt
      Menyimpan solusi optimal dari kelompok semut ini dalam bentuk objek dari kelas Ant.
    * antParty
      Menyimpan solusi-solusi lokal yang didapatkan oleh kelompok semut ini pada fase pelatihan / pencarian solusi terakhir.
    * antNumber
      Menyimpan informasi mengenai banyaknya semut (yang akan disebar saat proses pelatihan) pada kelompok semut ini.
  – Method
    * run
      Method ini merupakan method turunan dari interface Runnable. Method ini akan menjalankan method spreadAnt dengan atribut antNumber sebagai parameter yang dimasukkan. Setelah proses pencarian solusi selesai, method ini akan diberhentikan hingga proses pencarian solusi berikutnya dijalankan.
    * spreadAnt
      Method ini akan melakukan proses pelatihan / pencarian solusi berdasarkan nilai feromon pada atribut pheromone. Method ini akan membentuk solusi-solusi sebanyak nilai dari atribut integer yang dimasukkan. Solusi-solusi tersebut akan dibentuk dengan method constructPath. Setiap solusi yang dibentuk akan dicari nilai makespan dan nilai waittime dengan menggunakan kelas FlowShop yang disimpan pada atribut dan dibandingkan dengan solusi yang paling optimal pada saat ini. Solusi yang lebih baik akan disimpan pada atribut antOpt.

* constructPath
  Method ini akan membentuk salah satu urutan pengerjaan / solusi dari kasus flow shop yang ada pada atribut problem. Urutan pengerjaan / solusi tersebut di bentuk secara acak berdasarkan nilai-nilai feromon yang disimpan pada atribut pheromone. Method ini akan mengembalikan urutan pengerjaan / solusi dalam bentuk array kelas Job dengan indeks array sebagai nomor urutan pengerjaannya. Proses pembentukkan jalur dimulai dengan membuat duplikat array pekerjaan yang ada pada permasalahan flow shop. Proses selanjutnya akan menjalankan method pickRandomJob dengan memasukkan duplikat array pekerjaan tersebut dan angka -1. Angka -1 tersebut akan menunjukkan bahwa belum ada pekerjaan yang dipilih untuk dimasukkan ke dalam urutan pengerjaan.
  Method pickRandomJob akan memilih sebuah pekerjaan dari duplikat array pekerjaan secara acak berdasarkan nilai feromon yang disimpan. Pekerjaan yang dipilih dari oleh method pickRandomJob akan dimasukkan ke dalam urutan pengerjaan yang akan dihasilkan. Nomor / nama dari pekerjaan yang dipilih akan disimpan sebagai nomor pekerjaan terakhir dan pekerjaan dengan nomor / nama tersebut akan disingkirkan dari duplikat array pekerjaan dengan menggunakan method removeJob. Pemilihan urutan pekerjaan selanjutnya tetap dilakukan dengan method pickRandomJob. Parameter yang dimasukkan adalah array duplikat hasil dari method remove-
  Job dan nomor pekerjaan terakhir. Hasil method pickRandomJob ditambahkan pada hasil urutan pengerjaan. Nomor pekerjaan terakhir diubah sesuai dengan nomor pekerjaan hasil pickRandomJob dan pekerjaan dengan nomor tersebut disisihkan dari array duplikat. Proses ini dilakukan hingga seluruh pekerjaan pada array duplikat dikeluarkan.
* pickRandomJob
  Method ini akan memilih sebuah Job secara acak dari parameter array kelas Job yang dimasukkan. Pemilihan Job secara acak tersebut dipengaruhi oleh nilai feromon yang disimpan pada atribut pheromone. Method ini menerima 2 buah parameter : array kelas Job dan nomor penunjuk pekerjaan terakhir. Indeks matriks feromon yang mempengaruhi proses pemilihan pekerjaan ditentukan oleh nomor / nama pekerjaan yang terdapat pada parameter array dan nomor pekerjaan terakhir yang dimasukkan.
  Sebagai contoh, jika terdapat pekerjaan 2 pada parameter array kelas Job dan nomor pekerjaan terakhir adalah 3, maka indeks matriks [3][2] pada feromon akan mempengaruhi kemungkinan terpilihnya pekerjaan 2 di antara pekerjaan-pekerjaan pada parameter array kelas Job tersebut. Jika parameter nomor pekerjaan terakhir memiliki nilai -1, maka proses pemilihan akan menganggap belum ada nomor penunjuk pekerjaan terakhir (belum ada pekerjaan lain yang telah dikerjakan). Jika belum ada nomor penunjuk pekerjaan terakhir, maka proses pemilihan akan melibatkan jumlah dari nilai indeks matriks untuk masing-masing pekerjaan pada array kelas Job.
  Proses pemilihan pekerjaan dimulai dengan menjumlahkan nilai feromon dari indeks-indeks matriks yang mampu mempengaruhi pemilihan sebuah pekerjaan. Sebuah angka akan dibentuk secara acak di antara nilai 0 hingga jumlah nilai feromon tersebut. Proses selanjutnya akan melakukan proses pejumlahan nilai feromon dari indeks-indeks matriks yang mampu mempengaruhi pemilihan sebuah pekerjaan secara satu per satu. Jika penambahan nilai feromon dari suatu indeks matriks mengakibatkan nilai penjumlahan dari proses ini melebihi nilai angka acak dari proses sebelumnya, maka pekerjaan yang ditunjuk oleh indeks matriks tersebut merupakan pekerjaan yang terpilih.Pekerjaan tersebut kemudian akan diberikan

sebagai return value dari method ini. Pekerjaan tersebut akan diberikan dalam bentuk sebuah objek dari kelas Job.

* removeJobFrom
  Method ini menerima parameter sebuah array kelas Job dan sebuah angka penunjuk nomor / nama pekerjaan. Method ini akan menyisihkan pekerjaan dengan nomor / nama pekerjaan yang dimasukkan dari parameter array kelas Job. Method ini akan mengembalikan parameter array kelas Job tanpa objek Job dengan nomor / nama pekerjaan yang dimasukkan.
* getAntParty
  Method ini akan mengembalikan informasi mengenai solusi-solusi yang dibentuk oleh kelompok semut ini pada fase pelatihan / pencarian solusi terakhir.
* getAntOpt
  Method ini akan mengembalikan informasi mengenai solusi yang dianggap paling optimal hingga saat ini oleh kelompok semut ini.
* setAntOpt
  Method ini akan menginformasikan dan mengubah nilai dari solusi yang saat ini dianggap paling optimal oleh kelompok semut ini. Nilai dari solusi optimal tersebut akan diubah menjadi nilai solusi baru yang lebih optimal.

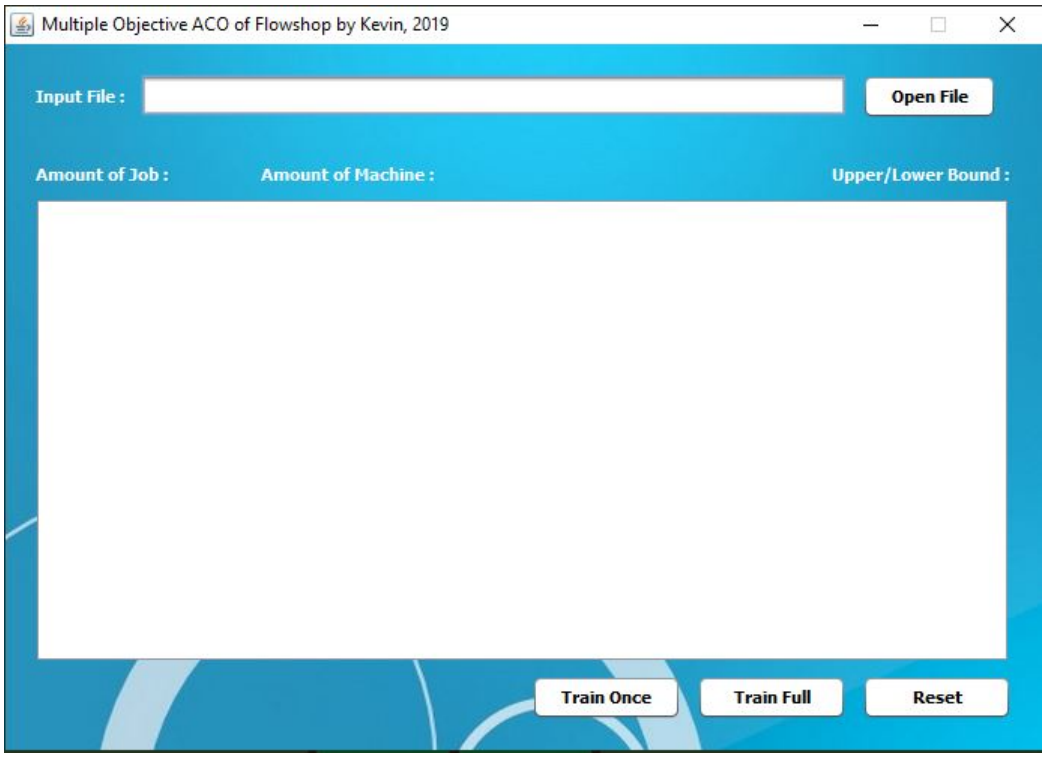
BAB 5

IMPLEMENTASI DAN PENGUJIAN

Pada bab ini, akan dijelaskan hasil implementasi algoritma pada perangkat lunak ini. Cara dan hasil pengujian perangkat lunak juga akan dijabarkan pada bab ini. Hasil pengujian tersebut akan digunakan pengukuran tingkat keoptimalan dari proses optimisasi yang dilakukan algoritma ant colony pada perangkat lunak.

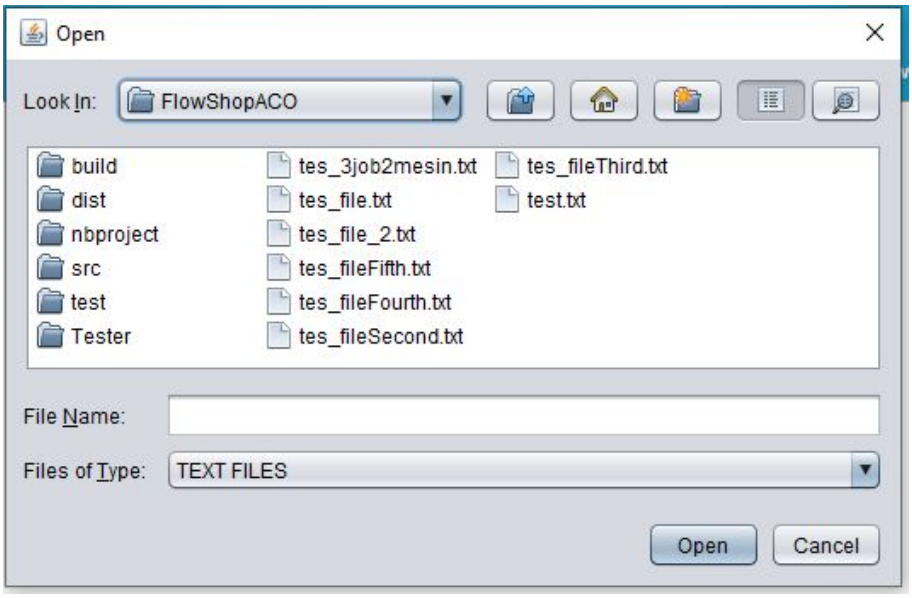
5.1 Implementasi Algoritma

Perangkat lunak ini akan mampu membentuk objek dari kelas AntColonyAlgorithm yang bertugas untuk mencari urutan pengerjaan / solusi yang optimal dari suatu kasus flow shop. Perangkat lunak ini akan mampu menerima data masukan sebuah kasus flow shop dari sebuah file teks dan membentuk objek AntColonyAlgorithm yang disesuaikan dengan data-data dari kasus yang diberikan. Tampilan awal dari perangkat lunak akan tampak seperti pada gambar 5.1



Gambar 5.1: Tampilan awal dari *interface* perangkat lunak

Pada tampilan awal ini, pengguna belum dapat menjalankan proses optimisasi. Untuk dapat menjalankan proses optimisasi, pengguna perlu memasukkan kasus flow shop terlebih dahulu melalui tombol *"Open File"*. Tombol *"Open File"* tersebut akan membuka *interface* baru seperti pada gambar 5.2 Interface tersebut berguna untuk memilih file teks yang berisi kasus flow shop.

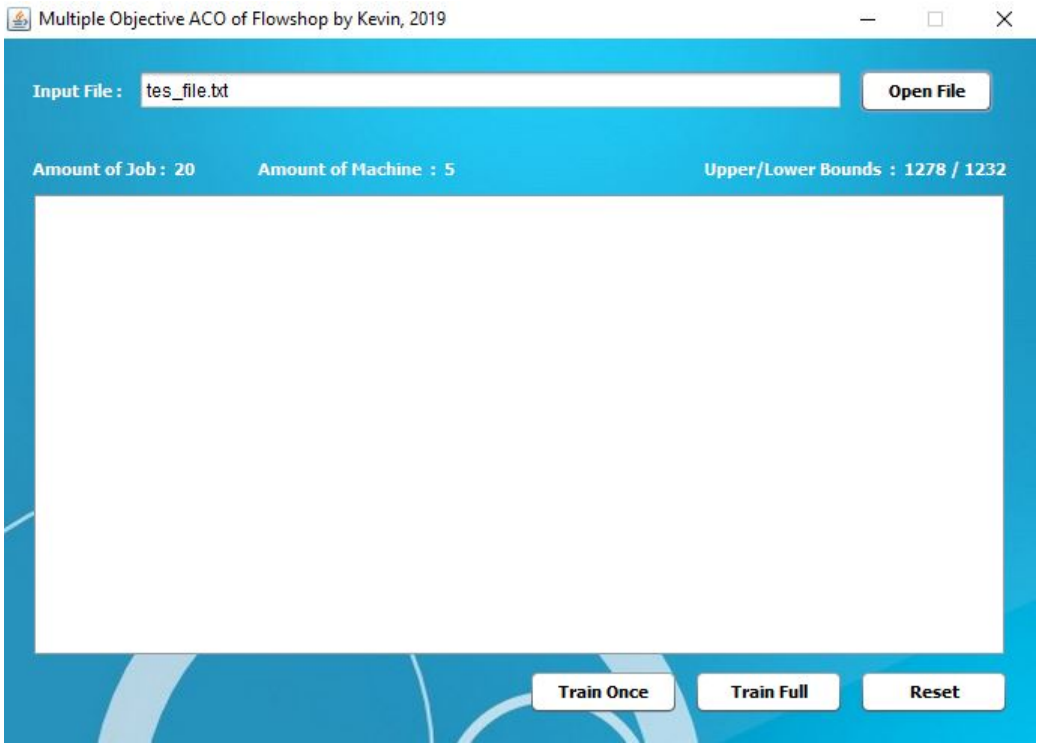


Gambar 5.2: Tampilan *interface* pemilihan kasus flow shop

Setelah kasus flow shop dipilih, perangkat lunak akan mengkonversi kasus tersebut menjadi objek dari kelas Problem dan membentuk objek AntColonyAlgorithm berdasarkan data-data kasus tersebut. Objek AntColonyAlgorithm tersebut akan membentuk objek dari kelas PheromoneDatabase dan AntColonySub yang akan membantu proses pencarian solusi optimal. Pembentukan objek dari kedua kelas tersebut akan diatur oleh objek AntColonyAlgorithm agar sesuai dengan kasus flow shop yang ingin dicari hasil optimalnya.

Objek dari kelas AntColonySub merepresentasikan sebuah kelompok semut yang akan disebar pada saat proses pelatihan algoritma ant colony. Objek-objek AntColonySub tersebut akan mampu melakukan pencarian solusi lokal secara bersama-sama. Banyaknya objek AntColonySub yang dibentuk akan disesuaikan dengan banyaknya pekerjaan pada kasus flow shop yang dimasukkan. Semakin banyak pekerjaan pada kasus flow shop yang diberikan, semakin banyak pula objek AntColonySub yang dibentuk.

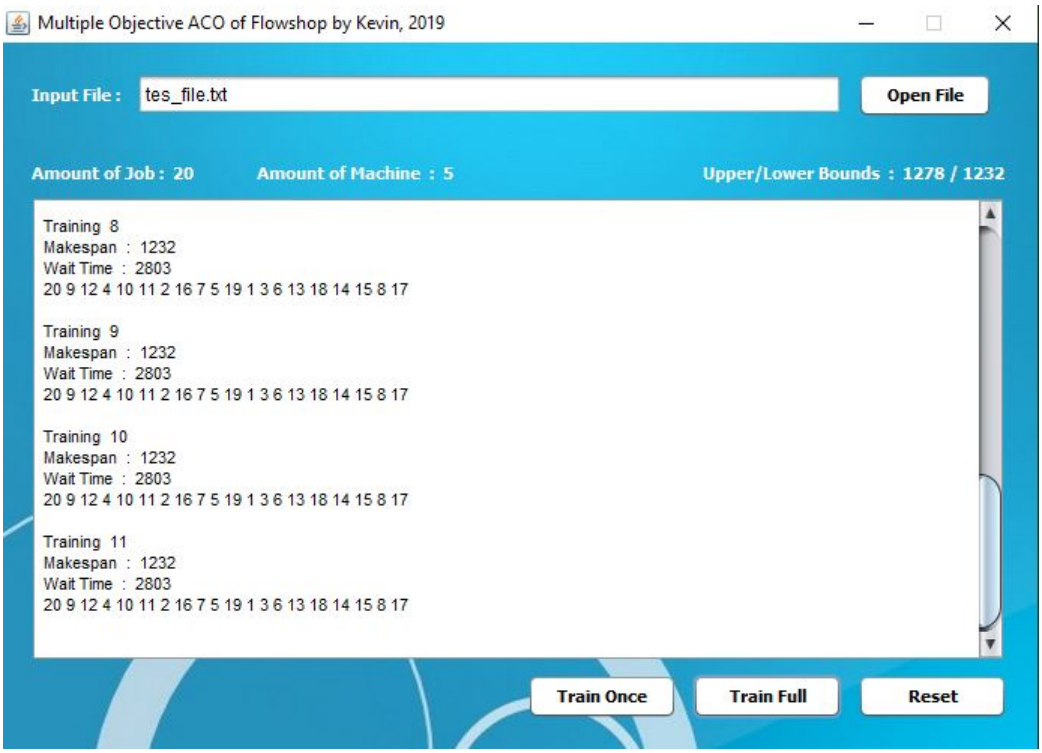
Setelah pengguna memasukkan kasus flow shop dan proses konversi kasus tersebut selesai, pengguna dapat mulai melakukan proses optimisasi. Informasi-informasi penting mengenai kasus flow shop yang ingin dicari hasil optimalnya akan ditampilkan pada interface perangkat lunak ini. Informasi mengenai banyaknya pekerjaan dan banyaknya mesin pada kasus flow shop akan ditampilkan pada *interface* program. Nilai *upper* dan *lower bound* juga akan ditampilkan di *interface* program. Tampilan interface tersebut akan tampak seperti gambar 5.3 .



Gambar 5.3: Tampilan *interface* setelah pemilihan kasus flow shop

Pengguna dapat menggunakan beberapa cara dalam melakukan proses optimisasi. Untuk melakukan proses optimisasi dengan menggunakan 1 kali fase pelatihan, pengguna dapat menekan tombol *"Train Once"*. Hasil dari proses optimisasi pada fase pelatihan ini akan ditampilkan pada text area. Nilai-nilai pada fase pelatihan ini akan dicatat oleh objek AntColonyAlgorithm dalam bentuk feromon dan akan mempengaruhi proses-proses optimisasi selanjutnya. Dengan dicatatnya fase-fase pelatihan yang dilakukan, hasil proses optimisasi akan cenderung lebih baik dan semakin mendekati solusi yang paling optimal.

Pengguna juga dapat melakukan proses optimisasi dengan melakukan fase pelatihan secara terus menerus hingga suatu kondisi berhenti tercapai. Proses optimisasi tersebut dapat dilakukan dengan menekan tombol *"Train Full"*. Setiap fase pelatihan yang terjadi pada proses ini akan mempengaruhi dengan fase pelatihan selanjutnya. Sama seperti fungsi pada tombol *"Train Once"*, hasil dari proses optimisasi pada setiap fase pelatihan yang terjadi akan ditampilkan pada text area.



Gambar 5.4: Tampilan interface setelah beberapa proses optimisasi

Tampilan *interface* setelah melakukan beberapa kali proses pelatihan / optimisasi akan terlihat seperti gambar 5.4 . Pengguna juga mampu mengulang proses optimisasi dari fase pelatihan pertama dengan menekan tombol "Reset". Pengguna juga dapat memasukkan kasus flow shop baru yang ingin dicari hasil optimalnya dengan menekan kembali tombol "Open File". Jika kasus flow shop yang baru telah dipilih, perangkat lunak akan membentuk ulang objek AntColonyAlgorithm agar sesuai dengan kasus baru yang dipilih dan memulai kembali proses optimisasi dari fase pelatihan pertama.

5.2 Pengujian Fungsional

Perangkat lunak ini akan memiliki beberapa fungsi, di antaranya :

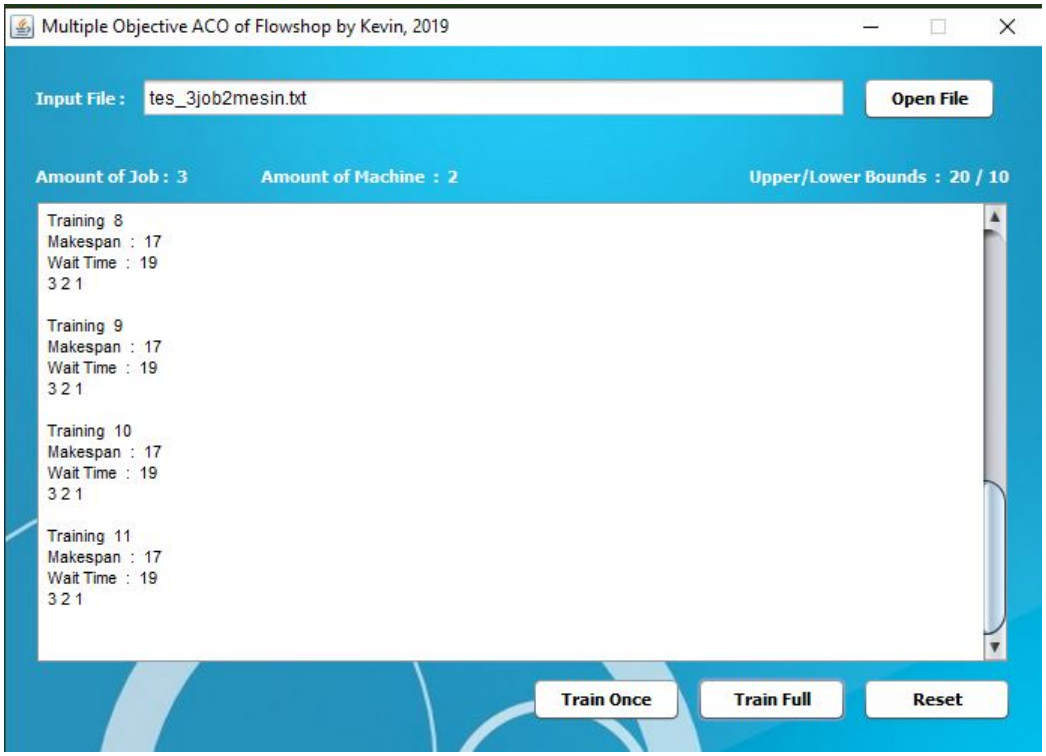
- Mengkonversi *file* teks menjadi kasus flow shop.
- Membentuk jalur secara acak.
- Menghitung nilai makespan dari kasus flow shop.
- Menghitung nilai wait time dari kasus flow shop.
- Membentuk feromon.
- Melakukan pelatihan dari algoritma ant colony.

Proses konversi *file* teks sudah dapat dianggap berhasil setelah perangkat lunak mampu menampilkan informasi banyaknya pekerjaan dan banyaknya mesin pada kasus flow shop. Pengujian proses perhitungan nilai makespan dan nilai wait time akan dilakukan dengan menggunakan kasus flow shop yang sederhana. Kasus flow shop yang digunakan melibatkan 3 buah pekerjaan dan 2 buah mesin. Kasus tersebut dijelaskan pada tabel 5.1 :

Tabel 5.1: Kasus Sederhana

	J1	J2	J3
Mesin 1	5	4	6
Mesin 2	2	1	7

Dari kasus tersebut kemudian dimasukan ke dalam *file* teks. Maka hasil optimasi dari kasus tersebut ada pada gambar 5.5



Gambar 5.5: Hasil Kasus Sederhana

Perangkat lunak yang dibuat sudah mampu menghasilkan nilai makespan 17 dan nilai wait time 19 dari kasus tersebut. Jalur secara acak juga sudah berhasil ditampilkan oleh perangkat lunak. Untuk membuktikan jika nilai makespan yang dihasilkan sudah benar maka akan dibuktikan lewat perhitungan pada gambar 5.6

	Waktu Proses		Mulai		Selesai	
Job	M1	M2	M1	M2	M1	M2
3	6	7	0	6	6	13
2	4	1	6	10	10	11
1	5	2	10	15	15	17

Gambar 5.6: Hasil Hitung Manual

Dari hasil tersebut nilai makespan yang dihasilkan adalah 17 sehingga sudah sesuai dengan hasil dari perangkat lunak. Sedangkan untuk nilai wait time sulit untuk dilakukan pemantauan secara langsung, pembuktian yang paling mungkin dilakukan adalah menganalisa lebih lanjut kode program pada kelas Flow Shop dimana terdapat *method* perhitungan wait time. Setelah proses pelatihan selesai, penambahan nilai feromon akan dilakukan oleh perangkat lunak. Nilai feromon yang dihasilkan oleh perangkat lunak ditunjukan pada tabel 5.2.

Tabel 5.2: Nilai Feromon

0	42	28
21	0	49
98	56	0

Tabel 5.2 merupakan tabel nilai feromon yang dihasilkan dari kasus flow shop dengan 3 buah pekerjaan. Pada tabel tersebut dapat dilihat bahwa terdapat beberapa indeks dengan nilai yang jauh lebih besar dibandingkan dengan indeks-indeks yang lain. Indeks-indeks feromon tersebut merupakan indeks-indeks feromon yang membantu pembentukan suatu solusi dan solusi tersebut merupakan solusi yang dianggap paling optimal pada saat ini.

5.3 Eksperimen

Pada bagian ini akan dijelaskan mengenai tata cara yang digunakan untuk melakukan eksperimen dan informasi apa saja yang diambil dari eksperimen tersebut. Hasil data dan analisa dari eksperimen tersebut juga akan dijelaskan.

5.3.1 Spesifikasi Perangkat Keras

Perangkat lunak ini dibangun menggunakan beberapa spesifikasi perangkat keras sebagai berikut :

1. Processor : Intel(R) Core(TM) i5-5200U 2.20GHz
2. Operating System : Windows 10 Pro 64-bit (10.0, Build 18362)
3. Memory: 4096MB RAM

5.3.2 Lingkungan Implementasi Perangkat Lunak

Perangkat lunak ini dibangun menggunakan beberapa spesifikasi perangkat lunak sebagai berikut :

1. NetBeans IDE versi 8.2
2. Bahasa pemrograman Java (jdk) versi 1.7

5.3.3 Cara Eksperimen

Eksperimen ini akan mencari tahu pengaruh jumlah semut, banyaknya pekerjaan, dan banyaknya mesin terhadap hasil optimisasi yang diberikan oleh algoritma ant colony. Eksperimen ini juga akan mencari tahu apakah algoritma ant colony ini sudah berjalan dengan benar. Data masukan yang akan digunakan berasal dari *benchmark tailard*. Data kasus yang digunakan dibatasi pada 20 pekerjaan dan 50 pekerjaan saja. Proses eksperimen akan dilakukan dengan memasukkan *file* teks dari beberapa jenis kasus flow shop yang disediakan oleh *tailard*. Masing-masing jenis kasus akan dibedakan berdasarkan banyaknya pekerjaan dan banyaknya mesin. Eksperimen akan melakukan optimisasi pada 10 kasus yang berbeda untuk setiap jenis kasus yang ada .

Proses eksperimen akan melakukan proses optimisasi untuk setiap data kasus sebanyak 5 kali. Hal ini dilakukan untuk mendapatkan rata-rata nilai makespan dan rata-rata nilai wait time yang didapatkan dari optimisasi sebuah data kasus. Pada eksperimen akan dilakukan optimisasi dengan menggunakan tombol *"Train Full"* secara berulang hingga fase pelatihan telah menghasilkan solusi yang sama sebanyak 10 kali. Pada setiap kasus, eksperimen akan melakukan optimisasi dengan 2 jumlah semut yang berbeda. Proses optimisasi ini akan melakukan proses pelatihan algoritma ant colony dengan melibatkan n dan n*2 semut, di mana n adalah banyaknya pekerjaan pada kasus flow shop yang dipilih. Jumlah semut tersebut digunakan dengan asumsi bahwa jumlah semut yang

disebar pada setiap fase pelatihan mempengaruhi tingkat kompleksitas dari algoritma ant colony. Semakin rumit kasus, maka semakin lama waktu yang dibutuhkan untuk proses optimisasi.

Hasil eksperimen yang akan dicatat adalah nilai makespan yang dihasilkan, urutan pengerjaan / solusi yang menghasilkan nilai makespan tersebut, nilai wait time yang dihasilkan, dan pada fase pelatihan ke berapa solusi tersebut didapatkan. Hasil dari nilai makespan yang diberikan akan dibandingkan dengan nilai *lower bound* dari *tailard*. Nilai *lower* itu sendiri akan dianggap sebagai nilai makespan target. Jika nilai makespan yang dihasilkan semakin mendekati nilai *lower bound*, maka semakin baik pula penilaian terhadap tingkat optimisasi algoritma ini.

5.3.4 Hasil Eksperimen

	N semut					N*2 semut				
	Makespan - Lower			Wait Time	Training	Makespan - Lower			Wait Time	Training
	Rata - Rata	Max	Min	Rata - Rata	Rata - Rata	Rata - Rata	Max	Min	Rata - Rata	Rata - Rata
20 JOB 5 MESIN	5.08	32	0	2601.8	12.6	1.75	11	0	2597.2	11.6
20 JOB 10 MESIN	6.19	19	0	3674.91	13.7	3.53	11	0	3673.998	13.44
20 JOB 20 MESIN	6.74	63	0	5757.24	14.5	2.39	21	0	5741.92	13.3
50 JOB 5 MESIN	4.55	13	0	6484.29	11.72	1.75	11	0	6464.04	11.92
50 JOB 10 MESIN	7.1	20	0	9761.73	12	1.76	9	0	9617.92	12.52
50 JOB 20 MESIN	7.25	25	0	13908.5	13.6	2.06	16	0	13872.27	13.46

Gambar 5.7: Hasil Eksperimen

Gambar 5.7 di atas berisi tabel data hasil eksperimen yang telah dilakukan. Untuk setiap kasus telah dilakukan perhitungan rata-rata selisih nilai makespan dengan nilai *lower bound* dari 5 kali pengulangan proses optimisasi. Nilai-nilai tersebut kemudian akan dikelompokkan berdasarkan banyaknya semut yang disebar dan jenis kasus flow shop yang menghasilkannya. Selain makespan dicatat juga nilai rata - rata wait time dari setiap kasus. Jenis dari suatu kasus dapat didefinisikan berdasarkan banyaknya pekerjaan dan mesin pada setiap tahap proses dari suatu kasus. Selain itu nilai rata - rata dari proses pelatihan untuk mendapatkan solusi juga dicatat pada eksperimen

5.3.5 Analisa Hasil Eksperimen

Berdasarkan hasil eksperimen, dapat dilihat bahwa algoritma ant colony mampu menghasilkan nilai makespan yang paling optimal. Hal ini dapat dilihat dari data minimal dari nilai minimal pada selisih makespan dengan *lower bound*. Nilai selisih 0 menunjukkan bahwa algoritma ant colony berhasil selalu mendapatkan nilai makespan yang paling optimal dari 5 kali proses optimisasi suatu kasus yang sama.

Nilai selisih makespan dengan *lower bound* akan semakin besar jika kasus yang ingin dioptimisasi semakin rumit. Pada optimisasi kasus flow shop dengan 50 buah pekerjaan, nilai selisih yang dihasilkan cenderung sangat besar. Jika dibandingkan dengan nilai selisih pada optimisasi kasus flow shop dengan 20 buah pekerjaan. Begitu juga dengan nilai wait time semakin rumit suatu kasus maka semakin tinggi pula nilai wait time yang dihasilkan. Semakin banyak pekerjaan yang ada pada kasus flow shop, maka tingkat keoptimalan dari hasil optimisasi algoritma ant colony ini akan semakin lama untuk dicapai.

Cara untuk menambah tingkat keoptimalan hasil optimisasi dapat dilakukan dengan menambah banyak semut yang disebar pada fase pelatihan. Hal ini dapat dilihat pada gambar 5.7 , pada hasil optimisasi kasus-kasus dengan jumlah semut yang berbeda. Hasil nilai selisih yang dihasilkan dengan **n*2** semut pada setiap fase pelatihan selalu bernilai lebih kecil jika dibandingkan dengan yang dihasilkan oleh **n** semut. Dari hal tersebut dapat dilihat bahwa semakin banyak semut yang disebar, maka hasil optimisasi yang dihasilkan juga akan cenderung lebih mendekati nilai yang paling optimal.

Banyaknya semut yang disebar juga mempengaruhi banyaknya fase pelatihan yang dibutuhkan untuk mencapai hasil optimal. Banyaknya fase pelatihan yang dilalui relatif lebih sedikit jika proses optimisasi dilakukan dengan jumlah semut yang lebih banyak. Di sisi lain, semakin rumit suatu kasus yang ingin dioptimisasi, maka semakin banyak pula fase pelatihan yang perlu dilalui untuk mencapai hasil yang optimal.

Cukup sulit untuk membuktikan bahwa algoritma ini telah bekerja sesuai dengan yang diinginkan. Algoritma ant colony sulit untuk dianalisa secara teoritis, karena membentuk solusi secara acak. Nilai-nilai feromon yang mempengaruhi pembentukan solusi tersebut juga selalu mengalami perubahan. Cara untuk menganalisa algoritma ini hanya dengan menggunakan cara eksperimental dengan mengambil beberapa sampel hasil solusi.

Tabel 5.3: Sampel hasil solusi

Nilai Makespan	Wait Time	Training	Urutan Pengerjaan
1185	2323	11	9 19 3 10 17 14 15 11 20 4 16 18 6 5 8 2 1 12 7 13
1183	2299	12	10 7 5 11 8 9 15 14 19 3 6 17 18 16 20 4 12 2 1 13
1181	2395	19	15 5 7 14 4 3 17 1 6 13 16 12 20 9 8 19 2 10 18 11
1185	2316	13	4 6 12 17 9 1 5 10 19 20 8 11 2 3 16 15 14 18 7 13
1180	2435	11	9 18 13 4 7 11 15 2 19 3 16 17 1 14 20 12 8 10 5 6

Tabel 5.3 di atas merupakan hasil pengujian salah satu kasus flow shop dengan 20 buah pekerjaan dan 5 mesin. Hasil optimisasi di atas dapat dijadikan data sampel untuk membuktikan bahwa algoritma ant colony yang diaplikasikan sudah bekerja dengan benar. Hal ini dapat dilihat dari dipilihnya pekerjaan 13 sebagai pekerjaan untuk dikerjakan terakhir pada tiga solusi yang didapatkan. Pekerjaan 9 juga cenderung untuk dikerjakan pertama kali.

Dalam satu kasus flow shop, algoritma ant colony akan cenderung mengarahkan dirinya untuk membentuk sebuah solusi yang sama. Solusi yang dijadikan tujuan tentu saja merupakan solusi yang paling optimal dari kasus flow shop tersebut. Pada perangkat lunak ini, algoritma ant colony akan berusaha membentuk urutan-urutan pekerjaan yang terbaik berdasarkan prioritas diambilnya suatu pekerjaan setelah diambilnya pekerjaan yang lain. Jika fase pelatihan dari hasil proses pengujian tersebut dilanjutkan, solusi yang dihasilkan kemungkinan besar akan semakin mirip bahkan dapat serupa.

BAB 6

KESIMPULAN DAN SARAN

Pada bab ini akan dijabarkan mengenai hal-hal apa saja yang dapat disimpulkan dari karya tulis ini. Beberapa saran untuk pengembangan selanjutnya dari perangkat lunak ini juga akan dijabarkan.

DAFTAR REFERENSI

[1] Baker, K. (1974) *Introduction to sequencing and scheduling.* Wiley.

[2] Bedworth, D. D. dan Bailey, J. E. (1999) *Integrated Production Control Systems: Management, Analysis, Design*, 2nd edition. John Wiley & Sons, Inc., New York, NY, USA.

LAMPIRAN A

KODE PROGRAM

Listing A.1: MyCode.c

```c

// This does not make algorithmic sense,
// but it shows off significant programming characters.

#include<stdio.h>

void myFunction( int input, float* output ) {
    switch ( array[i] ) {
        case 1: // This is silly code
            if ( a >= 0 || b <= 3 && c != x )
                *output += 0.005 + 20050;
            char = 'g';
            b = 2^n + ~right_size - leftSize * MAX_SIZE;
            c = (--aaa + &daa) / (bbb++ - ccc % 2 );
            strcpy(a,"hello $@?");
    }
    count = ~mask | 0x00FF00AA;
}

// Fonts for Displaying Program Code in LATEX
// Adrian P. Robson, nepsweb.co.uk
// 8 October 2012
// http://nepsweb.co.uk/docs/progfonts.pdf
```

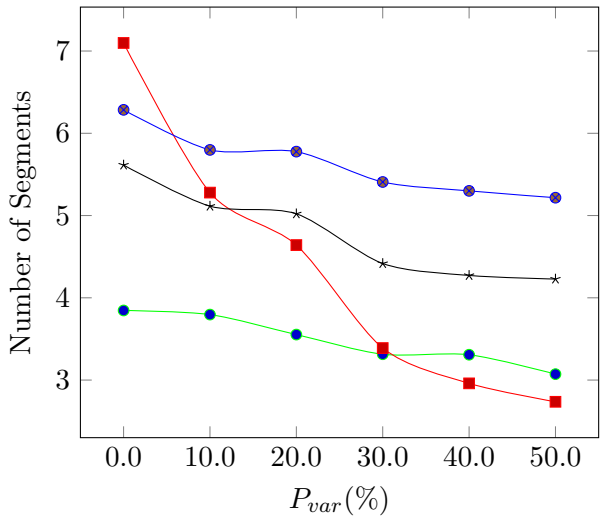
Listing A.2: MyCode.java

```java
import java.util.ArrayList;
import java.util.Collections;
import java.util.HashSet;

//class for set of vertices close to furthest edge
public class MyFurSet {
    protected int id;                                   //id of the set
    protected MyEdge FurthestEdge;                      //the furthest edge
    protected HashSet<MyVertex> set;                    //set of vertices close to furthest edge
    protected ArrayList<ArrayList<Integer>> ordered;    //list of all vertices in the set for each trajectory
    protected ArrayList<Integer> closeID;               //store the ID of all vertices
    protected ArrayList<Double> closeDist;              //store the distance of all vertices
    protected int totaltrj;                             //total trajectories in the set

    /*
     * Constructor
     * @param id : id of the set
     * @param totaltrj : total number of trajectories in the set
     * @param FurthestEdge : the furthest edge
     */
    public MyFurSet(int id,int totaltrj,MyEdge FurthestEdge) {
        this.id = id;
        this.totaltrj = totaltrj;
        this.FurthestEdge = FurthestEdge;
        set = new HashSet<MyVertex>();
        ordered = new ArrayList<ArrayList<Integer>>();
        for (int i=0;i<totaltrj;i++) ordered.add(new ArrayList<Integer>());
        closeID = new ArrayList<Integer>(totaltrj);
        closeDist = new ArrayList<Double>(totaltrj);
        for (int i = 0;i <totaltrj;i++) {
            closeID.add(-1);
            closeDist.add(Double.MAX_VALUE);
        }
    }

}
```

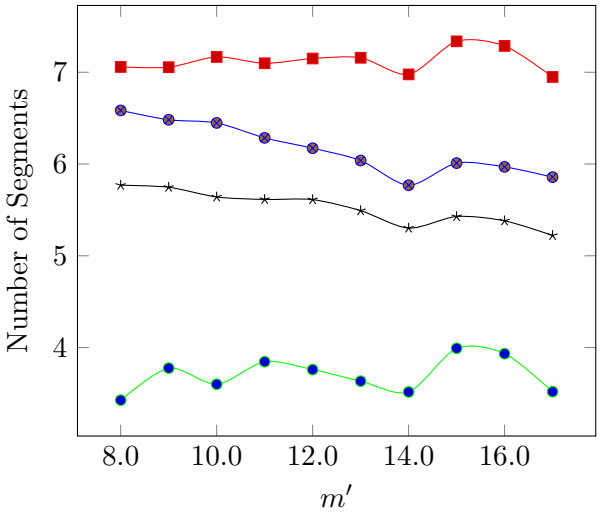
LAMPIRAN B

HASIL EKSPERIMEN

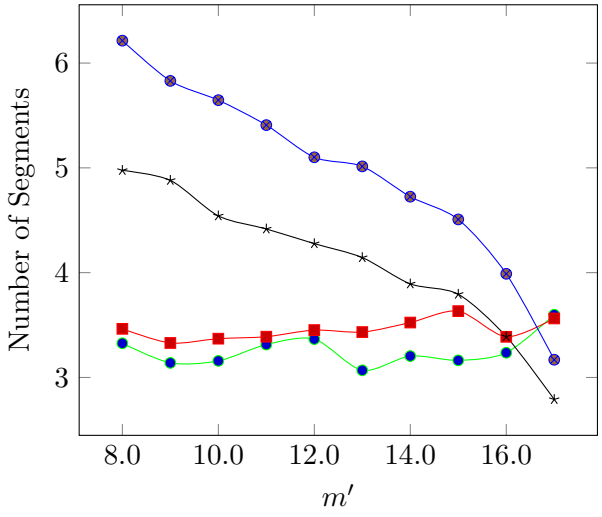
Hasil eksperimen berikut dibuat dengan menggunakan TIKZPICTURE (bukan hasil excel yg diubah ke file bitmap). Sangat berguna jika ingin menampilkan tabel (yang kuantitasnya sangat banyak) yang datanya dihasilkan dari program komputer.



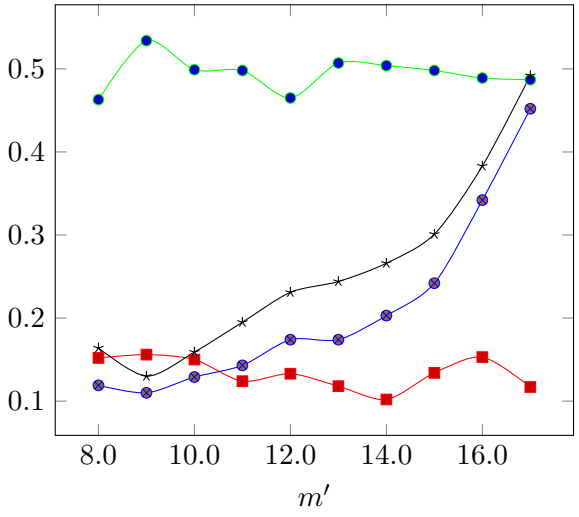
Gambar B.1: Hasil 1



Gambar B.2: Hasil 2



Gambar B.3: Hasil 3



Gambar B.4: Hasil 4