Tesis-TI.142307

MENYELESAIKAN *CONTAINER STOWAGE PROBLEM* (CSP) MENGGUNAKAN *ALGORITHM PARTICEL SWARM OPTIMIZATION* (PSO)

MATSAINI
2514 202 001

DOSEN PEMBIMBING
Ir. Budi Santosa, M.S., Ph.D

PROGRAM MAGISTER
OPTIMASI SISTEM INDUSTRI
JURUSAN TEKNIK INDUSTRI
FAKULTAS TEKNOLOGI INDUSTRI
INSTITUT TEKNOLOGI SEPULUH NOPEMBER
SURABAYA
2017

Tesis- TI.142307

COMPLETING THE CONTAINER STOWAGE PROBLEM (CSP) USING ALGORITHM PARTICEL SWARM OPTIMIZATION (PSO)

MATSAINI
2514 202 001

SUPERVISOR
Ir. Budi Santosa, M.S., Ph.D

MASTER PROGRAM
OPTIMIZATION SYSTEM OF INDUSTRY
INDUSTRIAL ENGINEERING DEPARTMENT
INDUSTRIAL TECHNOLOGY FACULTY
INSTITUT TEKNOLOGI SEPULUH NOPEMBER
SURABAYA
2017

MENYELESAIKAN *CONTAINER STOWAGE PROBLEM* (CSP) MENGGUNAKAN *ALGORITHM PARTICEL SWARM OPTIMIZATION* (PSO)

Tesis disusun untuk memenuhi salah satu syarat memperoleh gelar
Magister Teknik (MT)
di
Institut Teknologi Sepuluh Nopember Surabaya

Oleh :

MATSAINI
NRP. 2514 202 001

Tanggal Ujian : 5 Januari 2017
Periode Wisuda : Maret 2017

Disetujui oleh Tim Penguji Tesis:

1. Prof. Ir. Budi Santosa, MS., Ph.D. (Pembimbing)
   NIP. 196905121994021001

2. Prof. Dr. Ir. Suparno, MSIE (Penguji)
   NIP. 194807101976031002

3. Nurhadi Siswanto, S.T, MSIE., Ph.D. (Penguji)
   NIP. 197005231996011001

an. Direktur Program Pascasarjana
Asisten Direktur

Prof. Dr. Ir. Tri Widjaja, M.Eng.
NIP. 19611021 198603 1 001

Direktur Program Pascasarjana,

Prof. Ir. Djauhar Manfaat, M.Sc., Ph.D.
NIP. 19601202 198701 1 001

(Halaman ini sengaja dikosongkan)

SURAT PERNYATAAN KEASLIAN TESIS

Saya yang bertanda tangan dibawah ini:

Nama : MATSAINI

Program studi : Magister Teknik Industri – ITS

NRP : 2514202001

"**MENYELESAIKAN *CONTAINER STOWAGE PROBLEM* (CSP) MENGGUNAKAN *ALGORITHM PARTICEL SWARM OPTIMIZATION* (PSO)"**

Adalah benar-benar hasil karya intelektual mandiri, diselesaikan tanpa menggunakan bahan-bahan yang tidak diijinkan, dan bukan merupakan karya pihak lain yang saya akui sebagai karya sendiri.

Seluruh referensi yang dikutip dan dirujuk telah saya tulis secara lengkap di daftar pustaka.

Apabila dikemudian hari ternyata pernyataan saya ini tidak benar, maka saya bersedia menerima sanksi sesuai dengan peraturan yang berlaku.

Surabaya, 20 Januari 2017

Yang membuat pernyataan

MATSAINI

NRP. 2514202001

(Halaman ini sengaja dikosongkan)

MENYELESAIKAN *CONTAINER STOWAGE PROBLEM* (CSP) MENGGUNAKAN *ALGORITHM PARTICEL SWARM OPTIMIZATION* (PSO)

Nama Mahasiswa	: Matsaini
NRP	: 2514202001
Dosen Pembimbing	: Ir. Budi Santosa, M.S., Ph.D

ABSTRAK

Container Stowage Problem (CSP) adalah permasalahan penataan kontainer kedalam kapal dengan memperhatikan beberapa aturan penataan kontainer pada kapal seperti: total berat kontainer, berat satu tumpukan kontainer, tujuan kontainer, keseimbangan kapal, dan peletakan kontainer pada kapal, sehingga masalah penataan kontainer termasuk *Combinatorial Problems* yang susah dipecahkan dengan teknik *Enumerasi* dan termasuk NP-*Hard Problem* sehingga penyelesaian terbaik dengan metoda *heuristic.* Tujuan dari penelitian ini untuk meminimasi jumlah *shifting* sehingga diperoleh waktu *unloading* yang minimum. Dalam penelitian ini algoritma diusulkan adalah Modifikasi *Particle Swarm Optimization* (PSO) dengan menambahkan aturan perubahan posisi tumpukan, perubahan tumpukan berdasarkan tujuan, dan perubahan tumpukan berdasarkan jenis berat tumpukan (*Light, Medium,* dan *Heavy*). Algoritma usulan diaplikasikan pada lima macam kasus dan dibandingkan dengan algoritma *Modifikasi Bee Swarm Optimization.* Hasilnya algoritma PSO modifikasi lebih baik dari *Bee Swarm Optimization* (BSO) Modifikasi dengan nilai %Gap dan Gap bernilai negative yang artinya solusi dari PSO Modifikasi lebih kecil dari solusi BSO Modifikasi, perbandingan PSO Modifikasi terhadap solusi optimal dari Heuristik nilai rata-rata %Gap 0,87 persen dan Gap 60 detik, nilai ini lebih baik dari perbandingan BSO Modifikasi terhadap solusi optimal dari Heuristik dengan nilai rata-rata %Gap 2,98 persen dan Gap 459,6 detik.

Kata kunci: Container Stowage Problem, Particle Swarm Optimization.

(Halaman ini sengaja dikosongkan)

COMPLETING THE CONTAINER STOWAGE PROBLEM (CSP) USING ALGORITHM PARTICEL SWARM OPTIMIZATION (PSO)

Name	: Matsaini
NRP	: 2514202001
Supervisor	: Ir. Budi Santosa, M.S., Ph.D.

ABSTRACT

Container Stowage Problem (CSP) is the structuring of containers onto the ship with respect to some rules of the arrangement of containers on ships such as: total weight of the container, the weight of the pile of container, goal container, the balance of the ship, and the laying of containers on the ship, so the problem of structuring the container including Combinatorial Problems the trouble solving techniques Enumeration and included NP-Hard problem so the best solution with heuristic methods. The purpose of this study to minimize the amount of shifting in order to obtain the minimum unloading time. In this study, the proposed algorithm is Modified Particle Swarm Optimization (PSO) by adding a pile of position changes, changes in piles according to destination, and changes based on the type of heavy piles of piles (Light, Medium, and Heavy). The proposed algorithm was applied to the five kinds of cases and compared with the modification Bee Swarm Optimization algorithm. The result is a modified PSO algorithm is better than BSO Modifications to the value % Gap and Gap worth negative which means that the solution of Modified PSO smaller than Bee Swarm Optimization (BSO) solutions Modified, Modified PSO comparison to the optimal solution of a heuristic average % Gap and Gap value of 0.87 percent and 60 seconds, this value is better than the comparison BSO Modifications to the optimal solution of heuristics with an average % Gap and Gap value of 2.98 percent and 459.6 seconds.

Keywords: Container Stowage Problem, Particle Swarm Optimization.

(Halaman ini sengaja dikosongkan)

KATA PENGANTAR

Segala puji dan syukur Penulis panjatkan ke hadirat Allah SWT atas segala limpahan rahmat serta hidayahnya sehingga Penulis dapat menyelesaikan Tesis ini dengan judul **"MENYELESAIKAN *CONTAINER STOWAGE PROBLEM* (CSP) MENGGUNAKAN *ALGORITHM PARTICLE SWARM OPTIMIZATION* (PSO)"** dengan lancar. Taklupa shalawat dan salam kesejahteraan Penulis persembahkan ke haribaan sang pembawa perubahan dari zaman kegelapan menuju zaman terang benderang yaitu Nabi Muhammad SAW. Dalam menyelesaikan kasus *Container Stowage Problem* Penulis menggunakan agloritma PSO yang dimodifikasi oleh Penulis dengan manambahkan beberapa aturan agar mendapatkan fungsi tujuan yang minimum. Namun dalam proses penyelesaiannya Penulis banyak yang membantu demi terseselaikannya Tesis ini, baik yang membantu berupa pemikiran dan saran, serta berupa dukungan moril dan moral. Maka Penulis sangat pantas mengucapkan terimakasih banyak kepada:

1. Ayah (Matrawi) dan Ibu (Mistima) yang telah memberikan dukungan do'a, moran, maupun finansila.
2. Isri (Mayasrimartini, S.T.) yang selalu setia menemani dalam suasana suka ataupun duka dan memberikan semangat, dan anakku tercinta (Moh. Syafwan Ramadhan Al-Ghazali) yang menjadi motivasi buat Penulis. Dan juga adik-adik Penulis yaitu(Atnawi) dan (Qurratul Ainiyyah) yang telah memberikan semangat.
3. Bapak Prof. Ir. Budi Santosa, M.S., Ph.D selaku dosen pembimbing yang telah meluangkan waktu dan sabar dalam memberikan pengarahan dan pengetahuan selama proses pengerjaan Tesis ini.
4. Bapak Prof. Dr. Ir. Suparno, MSIE selaku dosen penguji pada persentasi pengajuan proposal Tesis dan sidang Tesis yang telah memberikan masukan pada penelititan ini sehingga terselesaikan Tesis ini.
5. Bapak Nurhadi Siswanto, ST, MSIE., Ph.D selaku dosen penguji sidang Tesis yang telah memberikan masukan pada penelititan ini sehingga terselesaikan Tesis ini.

6. Bapak Erwi Widodo, ST, M.Eng, Dr. Eng selaku Ketua Jurusan Program Pasca Sarjana Jurusan Teknik Industri Institut Teknologi Sepuluh Nopember dan dosen penguji pada persentasi pengajuan proposal Tesis yang telah memberikan masukan pada penelititan ini sehingga terselesaikan Tesis ini.
7. Seluruh dosen pengajar di Program Pasca Sarjana Jurusan Teknik Industri Institut Teknologi Sepuluh Nopember atas ilmu yang telah diberikan selama Penulis menempuh studi, serta seluruh staf dan karyawan di Jurusan Teknik Industri, terimakasih atas bantuannya dalam kepengurusan hingga Tesis ini selesai.
8. Seluruh rekan-rekan S2 TI ITS yang tidak cukup kalau disebutkan satu persatu disini, terimakasih atas kebersamaanya dan bantuannya.

Akhir kata dengan segala kerendahan hati Penulis minta maaf apabila ada kesalahan di dalam penulisan Tesis ini dan semoga Tesis ini dapat bermanfaat bagi para pembaca dan penelititan selanjutnya.

Surabaya, Januari 2017

MATSAINI

DAFTAR ISI

ABSTRAK v
ABSTRACT vii
KATA PENGANTAR ix
DAFTAR ISI xi
DAFTAR GAMBAR xiii
DAFTAR TABEL xv
BAB I PENDAHULUAN 1
1.1 Latar Belakang 1
1.2 Perumusan Masalah 5
1.3 Tujuan Penelitian 6
1.4 Ruang Lingkup Penelitian 6
1.4.1 Batasan 6
1.4.2 Asumsi 6
1.5 Manfaat Penelitian 7
1.6 Sistematika Penulisan 7
BAB II TINJAUAN PUSTAKA 9
2.1 *Container* Stowage Problem (CSP) 9
2.1.1 Ukuran dan tipe *container* 10
2.1.2 Berat *container* 11
2.1.3 Tujuan *container* 12
2.1.4 Keseimbangan kapal 12
2.1.5 Penamaan lokasi 14
2.2 Model CSP 15
2.3 Algoritma *Particle Swarm Optimization* (PSO) 18
2.4 Posisi Penelitian 20
BAB III METODOLOGI PENELITIAN 25
3.1 Metodologi Penelitian 25
3.2 Pengembangan algoritma 29
3.3 Contoh Numerik 38

3.3.1 Enumerasi 38
3.3.2 Pengembangan Algoritma *Particle Swarm Optimization* 39
3.4 Validasi 51
BAB IV EKSPERIMEN DAN ANALISIS 53
4.1 Pengujian Algoritma 53
4.1.1 Pengumpulan data 53
4.1.2 Implimentasi algoritma dan analisis. 54
4.2 Analisis validasi 76
4.3 Analisis *performance* algoritma 76
BAB V KESIMPULAN DAN SARAN 81
5.1 Kesimpulan 81
5.2 Saran 82
DAFTAR PUSTAKA 83
LAMPIRAN A 85
MATLAB CODE 85
LAMPIRAN B 111
DATA 111

DAFTAR GAMBAR

Gambar 2.1 Kecelakaan pada Tumpukan *Container* 12
Gambar 2.2 Tenggelamnya Kapal *Container*
Akibat Ketidak Seimbangan Muatan 13
Gambar 2.3 Cara Penamaan *Bay, Row,* dan *Tier* pada Kapal 14
Gambar 3.1 Penataan *Container* Berdasarkan Ukuran *Container* pada *Bay* 25
Gambar 3.2. *Flowchat* Metodologi Penelitian 28
Gambar 3.3 Penomoran *Bay, Row,* dan *Tier* 29
Gambar 3.4. *Flowchat* Pengembangan Algoritma
Particle Swarm Optimization 35
Gambar 3.5. Flowchat Pengembangan Algoritma
Particle Swarm Optimization (lanjutan) 36
Gambar 3.6. *Flowchat* Pengembangan Algoritma
Particle Swarm Optimization (lanjutan) 37
Gambar 3.7. Contoh Solusi Terbaik (dilihat dari atas) 39
Gambar 3.8. Contoh Jumlah Lokasi (dilihat dari atas) 40
Gambar 3.9 Penggabungan *Container* 20 *Feet* dan 40 *Feet* 46
Gambar 3.10. Hasil Penataan *Container*
pada Kasus Sederhana (dilihat dari atas) 47
Gambar 3.11. Hasil Penataan *Container* pada N1 (dilihat dari atas) 48
Gambar 3.12. Perbandingan Penataan Hasil Emunerasi dan PSO 52
Gambar 4.1 Grafik Rata-Rata Fungsi Tujuan 78
Gambar 4.2 %Gap untuk Semua Perbadingan Algoritma 79
Gambar 4.3 Gap untuk Semua Perbadingan Algoritma 79

(Halaman ini sengaja dikosongkan)

DAFTAR TABEL

Tabel 2.1. Tabel Penelitian Sebelumnya 23
Tabel 3.1. Contoh Kasus Sederhana 38
Tabel 3.2. Waktu *Unloading* 38
Tabel 3.3. Total Waktu *Unloading* pada Kasus Sederhana 49
Tabel 3.4. Total Penalti pada Kasus Sederhana 50
Tabel 3.5. Nilai Fungsi Tujuan pada Kasus Sederhana 50
Tabel 3.6. Perbandingan Nilai Fungsi Tujuan pada Enumerasi dan PSO 52
Tabel 4.1 Kriteria Data Untuk Setiap Kasus 54
Tabel 4.2 Perbandingan PSO dengan BSO Modifikasi pada Kasus 1. 55
Tabel 4.3 Perbandingan PSO Modifikasi dengan
BSO Modifikasi pada Kasus 1. 56
Tabel 4.4 Perbandingan PSO Modifikasi dengan
Solusi Optimal dari Metoda Heuristik pada Kasus 1. 57
Tabel 4.5 Perbandingan PSO Modifikasi dengan BSO Modifikasi
yang Paling Mendekati Solusi Optimal dari Metoda Heuristik pada Kasus 1. 58
Tabel 4.6 Perbandingan PSO dengan BSO Modifikasi pada Kasus 2. 59
Tabel 4.7 Perbandingan PSO Modifikasi dengan
BSO Modifikasi pada Kasus 2. 61
Tabel 4.8 Perbandingan PSO Modifikasi dengan Solusi
Optimal dari Metoda Heuristik pada Kasus 2. 62
Tabel 4.9 Perbandingan PSO Modifikasi dengan BSO Modifikasi
yang Paling Mendekati Solusi Optimal dari Metoda Heuristik pada Kasus 2. 63
Tabel 4.10 Perbandingan PSO dengan BSO Modifikasi pada Kasus 3. 64
Tabel 4.11 Perbandingan PSO Modifikasi dengan
BSO Modifikasi pada Kasus 3. 65
Tabel 4.12 Perbandingan PSO Modifikasi Dengan Solusi
Optimal dari Metoda Heuristik pada Kasus 3. 66
Tabel 4.13 Perbandingan PSO Modifikasi dengan BSO Modifikasi
yang Paling Mendekati Solusi Optimal dari Metoda Heuristik pada Kasus 3. 67

Tabel 4.14 Perbandingan PSO dengan BSO Modifikasi pada Kasus 4..... 68
Tabel 4.15 Perbandingan PSO Modifikasi dengan
BSO Modifikasi pada Kasus 4. 69
Tabel 4.16 Perbandingan PSO Modifikasi dengan Solusi
Optimal dari Metoda Heuristik pada Kasus 4. 70
Tabel 4.17 Perbandingan PSO Modifikasi dengan BSO Modifikasi yang Paling
Mendekati Solusi Optimal dari Metoda Heuristik pada Kasus 4. 71
Tabel 4.18 Perbandingan PSO dengan BSO Modifikasi pada Kasus 5..... 72
Tabel 4.19 Perbandingan PSO Modifikasi dengan
BSO Modifikasi pada Kasus 5. 73
Tabel 4.20 Perbandingan PSO Modifikasi dengan Solusi
Optimal dari Metoda Heuristik pada Kasus 5. 74
Tabel 4.21 Perbandingan PSO Modifikasi dengan BSO
Modifikasi yang Paling Mendekati Solusi Optimal dari
Metoda Heuristik pada Kasus 5..... 75
Tabel 4.22 Perbandingan Algoritma..... 78

BAB I
PENDAHULUAN

Bab pendahuluan ini meliputi latar belakang masalah, perumusan masalah, tujuan penelitian, ruang lingkup penelitian, dan manfaat penelitian.

1.1 Latar Belakang

Saat ini, lebih dari 70% perdagangan dunia menggunakan jalur laut, karena banyak kargo secara umum diangkut oleh kapal laut yang dimasukkan kedalam *container*, untuk menjadi perusahaan pelayaran yang kompetitif memilih untuk memenuhi *actual demand* dengan *container* yang lebih besar (Ambrosino, Paolucci, & Sciomachen, 2015). Transportasi laut merupakan hal penting dalam perdagangan internasional dan tidak diragukan lagi sebagai mesin pembangunan ekonomi global. Menurut ulasan *Maritime Transport* (2012) yang dirilis oleh Konferensi PBB mengenai perdagangan dan pembangunan, sekitar 80% dari banyaknya perdagangan global dilakukan melalui jalur laut. Diantara jenis kapal yang berbeda, sekitar 62% menggunakan *container*. Sejak dimulainya *containerization*, *container* memfasilitasi kelancaran arus barang dibeberapa alat transportasi tanpa penanganan terhadap barang secara langsung selama *shifting* (Wang, Jin, & Lim, 2015).

Seiring perkembangan penggunaan *container* yang semakin pesat membutuhkan suatu perencanaan yang tepat untuk menata *container* kedalam kapal. Perencanaan penataan *container (Container Stowage Planning)* adalah salah satu permasalahan yang kompleks yang dihadapi setiap hari oleh semua lini pengiriman (Fan et al, 2010). Maka dari itu, permasalahan penataan *container* kedalam kapal menjadi permasalahan yang rumit karena banyaknya *container* yang harus ditata dengan kapasitas kapal yang terbatas dan juga harus mempertimbangkan keseimbangan kapal karena akan melalui jalur laut yang begitu banyak rintangan, serta harus memperhatikan beberapa aturan seperti *container* dengan tujuan awal diletakkan di atas *container* dengan tujuan akhir, dan beberapa

aturan lainnya sehingga menjadi permasalahan yang memerlukan suatu solusi atau algoritma yang tepat untuk mengatasai permasalahan *Container Stowage Problem* (CSP) dengan tujuan meminimasi waktu *unloading* dan jumlah *shifting.*

Perencanaan yang baik dapat memaksimalkan dalam memamfaatkan ruang penyimpanan *container* pada kapal, namun perencanaan yang baik tidak mudah karena sangat tergantung pada pengalaman perencana (Fan et al., 2010). Dalam Penataan *container* kedalam kapal harus memperhatikan beberapa aturan seperti letak *container* harus sesuai dengan tipe dan ukuran *container, container* yang lebih berat diletakkan di bawah *container* yang lebih ringan, serta dalam proses penataan harus memperhatikan keseimbangan kapal. Penataan ribuan *container* yang memiliki puluhan tujuan dengan memperhatikan ukuran, tipe, berat dan keseimbangan kapal, permasalahan ini memunculkan banyak kemungkinan yang susah diselesaikan secara eksak (*combinatorial problems*). Dari sekian banyak kombinasi, ada kombinasi yang tidak mungkin bisa dilakukan sehingga ketika kombinasi itu dijalankan harus dilakukan *re-unloading* agar sebanyak mungkin *container* bisa masuk ke dalam kapal. Maka dari itu, dilakukan pendekatan heuristik untuk penataan *container* sehingga jumlah *re-unloading* atau *shifting* menjadi minimal (Putamawa & Santosa, 2011).

Penelitian tentang *container stowage problem* (CSP) telah banyak dilakukan sebelumnya dengan bermacam-macam teknik, diantaranya adalah *heuristics* (Ambrosino, Sciomachen, & Tanfani, 2004) menggunakan evaluasi eksak model 0-1 *Linear Programming*. Praktis untuk kasus sederhana, tapi tidak praktis untuk kasus besar. (Fan et al., 2010) *heuristic* dibangun dengan *Block Selection* yang terdiri dari dua *Stage Block Selection*, yaitu *Block Rangking* yaitu dilakukan perengkingan pada *Port Of Destination* (POD) dan *Block Allocation*. *Heuristics* (Monaco, Sammarra, & Sorrentino, 2014) melakukan *management* dengan tujuan meminimasi biaya yang berhubungan dengan *yard and transport operation*, mengusulkan *Binary Integer Programming* dan *Two-Step Heuristics Algorith*, langkah pertama adalah (CH) *Constructs an Initial Feasible Solution* dan langkah yang kedua (IH) *Search For Better Feasible Solution*. Kedua langkah tersebut didasarkan pada *Tabu Seach* yang merupakan *heuristics* pencarian lokal berulang dengan mekanisme memori. *Container* diklasifikasikan berdasarkan

atribut, seperti ukuran (*standard, 45-fouter, hight cube, over-sized*), berat (*light, medium, heavy*), tipe (*reefer, open- top*), beban (bahaya, mudah rusak), dan *port of destination* (POD). Komputasi menunjukkan efisiensi dan efektifitas. *Heuristics* (Ambrosino et al., 2015) menggunakan *Mixet Integer Programming* dibagi berdasarkan tipe kontainer, dan tiap-tiap tipe dibagi tiga kelas yaitu *light, medium, heavy clas*, tidak menemukan solusi yang layak dalam waktu satu jam. (Ding & Chou, 2015) dengan mengembangkan *Heuristics Algorithm* didapatkan hasil yang baik dibandingkan dengan *suspensory heuristics procedure (SH algorithm)*, tetapi hanya memuat *stowage planning* berdasarkan informasi pemuatan *port* saat ini saja, sedangkan algoritma SH memuat informasi dari semua *port.* Selain teknik *heuristics* kemudian banyak menggunakan algoritma *metaheuristics* seperti *Genetic Algorithm.*

Genetic Algorithm (Dubrovsky, Levitin, & Penn, 2002) menggunakan *compact encoding* untuk memperkecil pencarian, memberikan hasil yang memuaskan tapi jumlah iterasi sangat banyak. (Imai et al, 2006) dengan menambahkan *tournament* pada *Genetic Algorithm* untuk solusi yang lebih baik, tapi waktu komputasi menjadi lebih lama. *Genetic Algorithm* (Martins, Lobo, & Vairinhos, 2009) dengan memperhatikan keseimbangan *transverse* dan *longitudinal*, solusi baik tapi kasus yang digunakan bukan kasus yang kompleks dan besar. *A Bi-Level Genitic Algorithm* (Zhang et al, 2015) model didasarkan pada *Mixet Integer Programming* untuk dua permasalahan yaitu masalah *quay crane operating time* adalah untuk mengurangi dan mengoptimalkan bongkar muat, kemudian mengoptimalkan *stowage plan*. *A bi-level genitic algorithm* diusulkan untuk memecahkan model ini, dari percobaan numerik menunjukkan model dan algoritma lebih efektif dan tidak melebihi batas bawah, dan lebih baik jika dibandingkan dengan *Johnson's rule based heuristic algorithm method* (JHA) serta dengan metoda yang umum. Selain *Genetic Algorithm* ada beberapa algoritma *metaheuristics* yang diterapkan untuk menyelesaikan masalah CSP seperti, *Ant Colony Optimization, Branch and Bound,* dan *Tabu Seach.*

Ant Colony Optimization (Ambrosino et al, 2010) cocok untuk kasus yang sangat besar, sedangkan untuk kasus yang sedang lebih baik menggukanan *Tabu Search. Branch and Bound, Tabu Seach* (Wilson, Roach, & Ware, 2001) dibagi

dalam dua fase, *Strategic planning process* dengan *Branch and Bound* dan *tactical planning process* dengan *Tabu Seach*, solusi dapat tercapai tapi membutuhkan waktu cukup lama. *Contraint Programming and Integer Programming* (Delgado et al, 2012) dengan dua pendekatan yaitu *Constraint Programming* dan *Integer Programming*, keduanya menghasilkan hasil yang bagus. *Survey Method and Classification* (Lehnfeld & Knust, 2014) dilakukan pengedentifikasian pada kelas yaitu *Loading, Unloading, Premarshalling,* dan *Combained Problem*, kesimpulannya banyak permasalahan yang diselesaikan dengan cara eksak atau *heuristic methods* pada kelas *Loading* dan *Unloading. Mixet Integer Linear Programming* (Wang, Liu, & Meng, 2015) diaplikasikan pada *shiping network of global* yaitu permasalahan pengiriman *container* dengan kapal. Model yang dikembangkan diterapkan untuk jaringan pengiriman kapal Asia-Eropa-Oceania dengan total 46 port dan 11 rute kapal. Hasil menunjukkan bahwa masalah dapat diselesaikan secara efisien dan jaringan dioptimalkan mengurangi total biaya. *Hybrid Metaheuristic and Local Searc Heuristics* (Araujo et al, 2016) menggunakan *Clustering Search* merupakan metoda *Hybrid* dari *Metaheuristics* dan *Heuristics*.Dibagi empat konsep, *Search Metaheuristics* (SM) untuk menjelajahi ruang pencarian dengan memanipulasi beberapa solusi, *Iterative Clustering* (IC) untuk mengumpulkan solusi yang sama kedalam kelompok, *Analyzer Module* (AM) untuk memeriksa setiap klaster, dan *Local Searcher* (LS) yaitu pencarian internal. PCS lebih unggul dalam solusi untuk *mono-objective method*, untuk *Pareto fronts* unggul dalam semua kasus yaitu waktu dan solusi lebih baik.

Banyak penelitian terdahulu telah melakukan penelitian tentang permasalahan penataan *container* kedalam kapal karena *problem* penataan *container* ini termasuk NP-*Hard problem* sehingga penyelesaian terbaik dengan metoda *heuristic* (Avriel, Penn, & Shpirer, 2000). Maka dari itu peneliti melakukan penelitian tentang *Container Stowage Problem* dengan metoda *metaheuristic.* Algoritma yang digunakan adalah *Particle Swarm Optimization* (PSO) karena belum pernah dilakukan penerapan PSO untuk kasus *Container Stowage Problem* (CSP) dengan mempertimbangkan kelima factor yaitu tipe, ukuran, berat, tujuan, dan keseimbangan. PSO telah mampu diterapkan untuk menyelesaikan berbagai

persoalan optimasi yang termasuk NP-*Hard problem* seperti *vehicle routing problem* (VRP), penjadwalan proyek dengan *resources* terbatas (RCPSP), *traveling salesman problem* (TSP), *job shop scheduling* (Santosa dan Willy, 2011) .

Dalam kasus CSP algoritma PSO tidak bisa langsung diterapkan. Perlu adanya modifikasi terhadap PSO sehingga bisa diterapkan untuk menyelesaikan permasalahan CSP. Untuk permasalahan CSP dibuat suatu aturan untuk membangkitkan kordinat posisi suatu *container* dan aturan penataan *container*, kemudian dilaku perhitungan waktu *unloading* dan penalti dengan algoritma PSO, hasil yang didapatkan dari algoritma ini ketika kriteria penghentian telah terpenuhi seperti jumlah iterasi maksimum. Dalam penelitian ini peneliti menambahkan aturan pada PSO atau modifikasi PSO yaitu ketika hasil dari kriteria penghentian terpenuhi masih mungkin untuk dilakaukan perubahan terhadap posisi *container* berdasarkan jumlah *tier* atau masih banyak kordinat posisi dalam kapal yang yang tidak terisi sampai tingkat atas, maka setiap tumpukan posisi yang belum penuh sampai jumlah *tier* berpeluang untuk ditempatkan *container* baru yang didapatkan dari kordinat posisi lain dengan tumpukan posisi paling sedikit. Kemudian diurutkan berdasarkan tujuan *container*, berat *container*. Sehingga menghilangkan kedua kendala tersebut dalam menghitung *penalty*. Kehudian dihitung fungsi tujuan dan dibandingkan dengan fungsi tujuan sebelumnya, pilih nilai fungsi tujuan yang minimum. PSO juga termasuk algoritma yang simple yang didasarkan pada prilaku *swarm.* Percarian solusi dilakukan oleh suatu populasi yang terdiri dari beberapa partikel. Setiap partikel merepresentasikan posisi atau solusi dari permasalahan yang dihadapi.

1.2 Perumusan Masalah

Perumusan masalah dalam penelitian ini adalah penerapan algoritma *Particle Swarm Optimization* (PSO) untuk menyelesaikan *Container Stowage Problem* (CSP) dengan mempertimbangkan kelima faktor yaitu tipe, ukuran, berat, tujuan, dan keseimbangan. Dengan tujuan untuk memperolah jumlah *shifting* dan total waktu *unloading* yang minimum.

1.3 Tujuan Penelitian

Dalam penelitian ini tujuan yang ingin dicapai adalah:

1) Memperoleh algoritma *Particle Swarm Optimization* (PSO) untuk menyelesaikan *Container Stowage Problem* (CSP) dengan hasil yang lebih baik dari penelitian sebelumnya
2) Menghasilkan program dari *Particle Swarm Optimization* (PSO) untuk kasus *Container Stowage Problem* (CSP) sehingga bisa diaplikasikan untuk kasus yang nyata.

1.4 Ruang Lingkup Penelitian

Pada ruang lingkup penelitian ini dijelaskan mengenai batasan dan asumsi.

1.4.1 Batasan

1) Data yang digunakan merupakan data sekunder dari penelitian yang dilakukan Putamawa dan Santosa (2011), Putamawa dan Santosa membangkitkan data dengan bantuan software MATLAB dengan kriteria data dari penelitian Ambrosino et al (2004).
2) Hanya ada tiga tujuan *container*
3) Jenis *container* yang digunakan hanya ukuran 20 *feet* dan 40 *feet*.
4) Minimasi waktu hanya untuk proses *unloading* ketika *container* sampai pada tujuan pelabuhan ke satu, tujuan pelabuhan ke dua, dan tujuan pelabuhan ke tiga.

1.4.2 Asumsi

Asumsi yang digunakan dalam penelitian ini adalah jumlah *row* dan *tier* sama pada tiap *bay*.

1.5 Manfaat Penelitian

Manfaat dari penelitian ini adalah dapat mengaplikasikan algoritma *Particle Swarm Optimization* (PSO) sebagai pendekatan baru untuk menyelesaikan *Container Stowage Problem* dalam kasus yang nyata.

1.6 Sistematika Penulisan

Pada penulisan laporan penelitian ini dijelaskan secara sistematis sesuai dengan urutan kegiatan yang dilakukan peneliti dalam menyelesaikan permasalahan. Sistematika penulisan sebagai berikut:

BAB I. PENDAHULUAN

Pada bab ini berisi penjelasan tentang hal-hal yang melatarbelakangi dilakukannya penelitian, kemudian dilakukan perumusan masalah dengan tujuan yang ingin dicapai dari penelitian yang akan dilakukan, ruang lingkup penelitian yang terdiri dari batasan dan asumsi yang digunakan, dan mamfaat penelitian, serta sistematika penulisan dalam penelitian ini.

BAB II. TINJAUAN PUSTAKA

Tinjauan pustaka menguraikan teori dari permasalahan dan metoda yang digunakan. Teori diperoleh dari referensi yang digunakan sebagai landasan untuk melakukan kegiatan penelitian.

BAB III. METODOLOGI PENELITIAN

Metodologi penelitian digunakan sebagai pedoman agar penelitian dapat berjalan secara terstruktur sesuai kerangka penelitian, dan menguraikan metodologi penelitian yang digunakan serta tahapan pengembangan algoritma untuk menyelesaikan permasalahn yang dihadapi.

BAB IV. EKSPERIMEN DAN ANALISIS

Dalam bab eksperimen dan pembahasan akan dilakukan analisis terhadap hasil uji coba algoritma yang telah dilakukan sebelumnya. Analisis dilakukan untuk setiap pengujian data yang digunakan.

BAB V. KESIMPULAN DAN SARAN

Bab ini berisi tentang kesimpulan hasil penelitian dan saran-saran yang berkaitan dengan penelitian selanjutnya.

BAB II
TINJAUAN PUSTAKA

Bab tinjauan pustaka mengurai tentang landasan teori, temuan peneliti pendahulu yang berhubungan dengan topik penelitian yang dijadikan acuan sebagai landasan dilakukannya penelitian.

2.1 *Container* Stowage Problem (CSP)

Pengelolaan *container* pada terminal merupakan proses yang kompleks yang melibatkan banyak keputusan yang saling terkait. Pengangkutan *container* berbagai macam cara misalnya, dengan truk, kereta api, dan kapal. Maka terminal merupakan dasar dalam jaringan transportasi, dan untuk alasan ini semua operasi yang terlibat dalam arus *container* harus dioptimalkan untuk mencapai produktivitas global yang maksimum, dinyatakan dalam beberapa indikator ekonomi yang tepat. Karena biaya tinggi yang terkait dengan total waktu yang dihabiskan oleh sebuah kapal di terminal, semua perusahaan maritim merujuk indikator produktivitas, misalnya yang berkaitan dengan jam operasi, dan pemilihan rute kapal atau urutan pelabuhan untuk dikunjungi (Ambrosino et al., 2004). *Container Stowage Problem* (CSP) adalah permasalahan penataan *container* kedalam kapal yang sering desebut dengan *Master Bay Plan Problem* (MBPP). Permasalahan penataan ini yang selalu dihadapai sehiri-hari oleh masing-masing *terminal management* (Ambrosino et al., 2010). Sebelumnya masalah penataan *container* kedalam kapal dilakukan oleh kapten kapal, seiring berkembangnya penataan *container* yang banyak dan tidak mungkin dilakukan dengan metoda *try and error* karena akan banyak menghabiskan tenaga dan uang demi penataan *container* yang tepat kedalam kapal. Maka banyak peneliti menggukanan teknik optimasi untuk menyelesaikan permasalahan penataan *container*. karena jumlah *container* yang diangkut terus meningkat, maka semakin banyak alternatif yang muncul sehingga kemudian muncul *Container Stowage Problem* yang merupakan *Combinatorial Problem* dan NP-*Hard Problem* (Putamawa & Santosa, 2011).

Permasalahan penataan *container* kedalam kapal dikatakan permasalahan *Combinatorial Problem* dan NP-*Hard Problem* karena harus memenuhi aturan pada kapal dan meminimalkan terjadinya *shifting*, dan penataan harus tepat agar *container* tidak rusak akibat tertindih oleh *container* yang lain. Maka dalam melakukan penataan *container* harus memenuhi aturan seperti peletakan *container* harus sesuai dengan ukuran dan tipe *container*, berat *container* yaitu *container* yang lebih berat berada dibawah *container* yang lebih ringan, tujuan *container* yaitu tujuan yang lebih awal diletakkan diatas *container* dengan tujuan yang lebih akhir, dan total berat *container* harus sesuai dengan keseimbangan kapal agar kapal tidak mudah tenggelam.

2.1.1 Ukuran dan tipe *container*

Ukuran *container* ada 5 macam yaitu 20 *feet* dengan panjang 6.1 m, 40 *feet* dengan panjang 12.2 m, 45 *feet* dengan panjang 13.7 m, 48 *feet* dengan panjang 14.6 m, dan 53 *feet* dengan panjang 16.2 m, dengan ukuran lebar yang sama, yaitu 8 *feet*. Namun, *container* yang sering digunakan pada transportasi laut adalah *container* dengan panjang 20 *feet* dan 40 *feet*. *Container* 20 *feet* biasanya disebut dengan satu TEU (*Twenty-foot Equivalent Units*) dan *container* 40 *feet* disebut dengan dua TEU atau satu FEU (*Forty-foot Equivalent Unit*). Pada proses *stowage*, kedua *container* tersebut dipisahkan karena diletakkan pada tempat yang berbeda. Masing-masing ukuran *container* memiliki tempat sendiri di kapal. Meskipun FEU memiliki ukuran dua kali TEU, FEU tidak bisa diletakkan pada dua lokasi TEU dan sebaliknya, satu lokasi FEU tidak bisa ditempati dua buah TEU. Hal tersebut dilakukan untuk mempermudah jika terjadi *shifting*.

Selain dibedakan berdasarkan ukuran, *container* juga dibedakan berdasarkan tipenya. Macam-macam tipe *container* adalah sebagai berikut:

- *General/ Dry container* adalah *container* biasa seperti pada umumnya.
- *Refrigerator container* adalah *container* yang dilengkapi dengan sistem pendingin.
- *Hazardous container* adalah *container* yang digunakan untuk memuat barang yang mudah meledak atau berbahaya.

- *Tank container* adalah *container* berbentuk tangki yang digunakan untuk memuat barang cair.

Dalam proses *stowage, container* tersebut dipisah karena memiliki tempat khusus, kecuali *general/dry container*. *Refrigerator container* memerlukan sumber listrik untuk menyalakan sistem pendingin sehingga harus diletakkan di area yang memiliki sumber listrik. *Hazardous container* harus dipisahkan dari *container* lain karena rawan meledak dan radiasi. *Tank container* yang berbentuk tangki harus diletakkan paling atas karena di atasnya tidak boleh ditumpuk *container*.

2.1.2 Berat *container*

Setiap *container* memiliki ukuran lebar yang sama dan panjang yang dikelompokkan menjadi 5 macam ukuran. Walaupun dalam satu ukuran, berat *container* berbeda-beda karena memuat barang yang berbeda-beda. Secara umum berat *container* dikelompokkan menjadi tiga jenis, yaitu *light* (5-15 ton), *medium* (15-25 ton), dan *heavy* (lebih dari 25 ton) (Ambrosino et al., 2004). Dalam penataan *container*, berat *container* sangat diperhatikan, karena mempengaruhi keseimbangan tumpukan *container*, maka *container* yang lebih berat harus diletakkan dibawah *container* yang lebih ringan untuk menghindari terjadinya kerusakan pada *container* akibat tidak kuat menaham beban yang diatasnya sehingga tumpukan menjadi goyah dan roboh. Seperti yang terlihat pada Gambar 2.1 robohnya tumpukan *container* diakibatkan *container* yang dibawah tidak kuat menahan beban yang diatasnya karena lebih berat, sehingga *container* yang dibawah menjadi goyah dan tumpukan menjadi roboh.

Gambar 2.1 Kecelakaan pada Tumpukan *Container*

2.1.3 Tujuan *container*

Tujuan Kapal pengangkut *container* tidak hanya berlayar ke satu tujuan, tapi lebih dari satu tujuan, sehingga *container* yang diangkut memiliki bermacam-macam tujuan. Setelah mencapai tujuan, *container* akan di-*unstowage* sedangkan *container* yang lain tetap berada di kapal. Tapi, ketika *container* yang akan di-*unstowage* terhalang *container* lain yang seharusnya tetap di kapal, maka perlu dilakukan *shifting* (pemidahan sementara). Untuk mengurangi jumlah *shifting*, dalam proses *stowage* diusahakan agar *container* dengan tujuan terdekat terletak paling atas dan tujuan terjauh terletak paling bawah (Ambrosino et al., 2004).

2.1.4 Keseimbangan kapal

Pengiriman *container* dengan kapal sangat penting untuk menjaga keseimbangan, penataan *container* yang kurang tepat akan menyebabkan keseimbangan kapal menjadi tidak seimbang. Maka dalam menataan *container* kedalam kapal harus memperhatikan tiga macam keseimbangan pada kapal, yaitu.

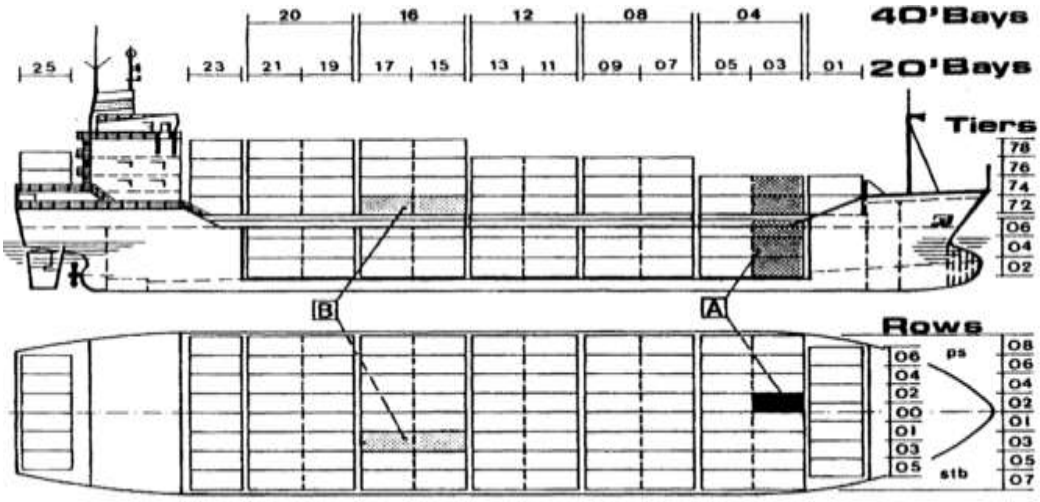
- *Cross equilibrium*, yaitu keseimbangan antara bagian kanan dan kiri kapal. Selisih berat bagian kanan dan bagian kiri tidak boleh melebihi batas toleransi yang telah ditentukan.
- *Horizontal equilibrium*, yaitu keseimbangan antara bagian depan dan belakang. Selisih berat bagian depan dan bagian belakang juga tidak boleh melebihi batas toleransi yang telah ditentukan.
- *Vertical equilibrium*, yaitu keseimbangan antara bagian atas (*upper deck*) dengan bagian bawah (*lower deck*). Untuk kapal yang memiliki *upper deck* dan *lower deck*, maka berat pada bagian *upper deck* harus lebih ringan atau sama dengan berat pada bagian lower deck.

Dari ketiga macam keseimbangan tersebut harus terpenuhi ketika menata *container* kedalam kapal untuk menghindari terjadinya kecelakaan pada kapal *container*, seperti kapal tenggelam akibat tidak seimbangnya muatan seperti yang terlihat pada Gambar 2.2 yaitu tenggelamnya kapal *container* akibat ketidak seimbangan muatan.

Gambar 2.2 Tenggelamnya Kapal *Container* Akibat Ketidak Seimbangan Muatan

2.1.5 Penamaan lokasi

Setiap lokasi *container* pada kapal memiliki indek posisi atau keterangan lokasi pada kapal, yaitu posisi *bay*, *row*, dan *tier*. Penamaan *bay* dihitung dari bagian depan kapal ke bagian belakang kapal. Sedangkan penamaan *row* dihitung dari bagian tengah kapal ke bagian luar kapal jika kapal dilihat dari atas. Untuk penamaan *tier* dihitung dari bagian bawah ke bagian atas jika kapal dilihat dari samping. Contoh penamaan *bay*, *row*, dan *tier* dapat dilihat pada Gambar 2.3.



Gambar 2.3 Cara Penamaan *Bay*, *Row*, dan *Tier* pada Kapal

Pada Gambar 2.3 dapat dilihat penamaan untuk setiap lokasi berdasarkan nomor *bay*, *row*, dan *tier*. Penamaan *bay* untuk *container* ukuran 20 diletakkan pada *bay* ganjil yaitu *bay* 01, 03, 05, 07, 09 dan seterusnya dan disebut *bay* ganjil. Sedangkan penamaan *bay* untuk *container* ukuran 40 diletakkan pada *bay* genap atau dua *bay* ganjil yaitu *bay* 04 = *bay* 03 + *bay* 05, *bay* 08 = *bay* 07 + *bay* 09 dan seterusnya dan disebut *bay* genap (Ambrosino et al., 2004). Penomoran *container* ukuran 20 atau *bay* ganjil dimulai dari angka 01 berlanjut dengan penambahan 2 angka pada nomor *container* dibelakangnya, dan penomoran *container* ukuran 40 atau *bay* genap dimulai dari angka 02 jika *container* ukuran 40 bisa diletakkan pada

urutan pertama atau angka 04 jika *container* ukuran 40 tidak bisa diletakkan pada urutan pertama berlanjut dengan penambahan 4 angka pada nomor *container* dibelakangnya. Sedangkan penomoran untuk *row* dimulai dari bagian tengah, *row* pada bagian kanan diberi nomor ganjil dan *row* bagian kiri diberi nomor genap, penomoran *row* dengan jumlah *row* ganjil dimulai dari nomor 00 pada posisi *amitships* atau titik tengah pada *row*, pada *row* bagian kanan dimulai dari 01, 03, 05 dan seterusnya, pada *row* bagian kiri dimulai dari 02, 04, 06 dan seterusnya. Jika jumlah *row* genap penomoran *row* dimulai dari 01, 02 pada posisi *amitships*, pada *row* bagian kanan dimulai dari 01, 03, 05, 07 dan seterusnya, pada *row* bagian kiri dimulai dari 02, 04, 06, 08 dan seterusnya. Dan penomoran pada *tier* dimulai dari bagian bawah kebagian atas, dimulai dari angka 02, 04, 06 berlanjut dengan penambahan 2 angka pada nomor *container* dibawahnya.

2.2 Model CSP

Banyak penelitian tentang *Container Stowage Problem* dengan berbagai macam metoda dan beberapa model. Satu diantara model tersebut yang memperhatikan kelima faktor dalam proses *stowage* (ukuran, tipe, berat, dan tujuan *container*, serta keseimbangan kapal) adalah model yang dibangun Ambrosino, Sciomachen, dan Tanfani (2004). Model dasar ini juga yang pernah dijadikan rujukan penelitian oleh Putamawa dan Santosa (2011). Model dasar tersebut adalah:

$$Min\ L = \sum_l \sum_c t_{lc} x_{lc} \quad (2.1)$$

$$\sum_l \sum_c x_{lc} = m \quad (2.2)$$

$$\sum_l x_{lc} \leq 1\ \forall c \quad (2.3)$$

$$\sum_c x_{lc} \leq 1\ \forall l \quad (2.4)$$

$$\sum_l \sum_c w_c x_{lc} \leq Q \quad (2.5)$$

$$\sum_{c \in T} x_{ijkc} = 0 \quad \forall i \in E, j, k \quad (2.6)$$

$$\sum_{c \in F} x_{ijkc} = 0 \quad \forall i \in O, j, k \quad (2.7)$$

$$\sum_{c \in T} x_{i+1jkc} + \sum_{c \in F} x_{i+1jkc} \leq 1 \quad \forall i \in E, j, k \quad (2.8)$$

$$\sum_{c \in T} x_{i-1jkc} + \sum_{c \in F} x_{i+1jkc} \leq 1 \quad \forall i \in E, j, k \quad (2.9)$$

$$\sum_{c\in T} x_{i+1jk+lc} + \sum_{c\in F} x_{i+1jkc} \leq 1 \quad \forall i \in E, j, k = 1, \dots, |k| - 1 \tag{2.10}$$

$$\sum_{c\in T} x_{i-1jk+lc} + \sum_{c\in F} x_{i+1jkc} \leq 1 \quad \forall i \in E, j, k = 1, \dots, |k| - 1 \tag{2.11}$$

$$\sum_{k} \sum_{c\in T} w_c x_{ijkc} \leq MT \quad \forall i, j \tag{2.12}$$

$$\sum_{k} \sum_{c\in T} w_c x_{ijkc} \leq MF \quad \forall i, j \tag{2.13}$$

$$\sum_{\substack{c,e\in C: \\ w_c \neq w_e}} \left(w_c x_{ijkc} - w_e x_{ijk+1e}\right) \geq 0 \quad \forall i, j, k = 1, \dots, |k| - 1 \tag{2.14}$$

$$\sum_{\substack{c,e\in C: \\ d_c \neq d_e}} \left(d_c x_{ijkc} - d_e x_{ijk+1e}\right) \geq 0 \quad \forall i, j, k = 1, \dots, |k| - 1 \tag{2.15}$$

$$-Q_2 \leq \sum_{i\in A,j,k} \sum_c w_c x_{ijkc} - \sum_{i\in P,j,k} \sum_c w_c x_{ijkc} \leq Q_2 \tag{2.16}$$

$$-Q_1 \leq \sum_{i,j\in L,k} \sum_c w_c x_{ijkc} - \sum_{i,j\in R,j,k} \sum_c w_c x_{ijkc} \leq Q_1 \tag{2.17}$$

$$x_{lc} \in \{0,1\} \quad \forall l, c \tag{2.18}$$

Fungsi tujuan dari model matematis ini meminimalkan total waktu *unloading* (L), dengan variabel keputusannya adalah:

x_{lc} = Variabel untuk *container* ke-*c* di lokasi *l*, sama dengan 1 jika *container* ke-*c* berada di lokasi l dan sama dengan 0 jika tidak.

$x_{lc} = x_{ijk}$

Untuk keterangan variabel lain yang digunakan dalam model matematis tersebut adalah sebagai berikut:

l = Lokasi penempatan ke – 1,...., n yang ditunjukkan dengan i = *bay*, j = *row*, dan k = *tier* (l = 1,2,…,n)

c = *Container* ke – 1,...,m

t_{lc} = Waktu yang diperlukan untuk *unloading container* ke-*c* di lokasi *l*

m = Jumlah semua *container*

w_c = Berat *container* ke-*c*

Q = Total kapasitas

Q1 = Toleransi keseimbangan *anterior-posterior*

Q2 = Toleransi keseimbangan *left-right*

E = *Bay* genap

O = *Bay* ganjil

T = *Container* dengan ukuran 20 *feet*

F = *Container* dengan ukuran 40 *feet*

MT = Batas berat satu tingkat *container* 20 *feet*

MF = Batas berat satu tingkat *container* 40 *feet*

K = *Tier* kapal

d_c = Tujuan dari *container*

Untuk keterangan tiap persamaan pada model matematis adalah sebagai beriku:

(1) Fungsi tujuan, yaitu meminimasi total waktu *unloading*

(2) Total lokasi l yang ditempati *container* harus sama dengan jumlah *container*.

(3) Satu *container* hanya ditempatkan pada satu lokasi.

(4) Satu lokasi hanya bisa ditempati satu *container*.

(5) Total berat *container* yang diangkut tidak boleh melebihi kapasitas kapal (Q).

(6) *Container* 20 *feet* (T) tidak akan ditempatkan pada *bay* genap (E). Sesuai dengan peraturan penataan berdasarkan ukuran *container*, yaitu *container* 20 *feet* diletakkan pada *bay* ganjil.

(7) *Container* 40 *feet* (F) tidak akan ditempatkan pada *bay* ganjil (O). Sesuai dengan peraturan penataan berdasarkan ukuran *container*, yaitu *container* 40 *feet* diletakkan pada *bay* genap.

(8) dan (9) *Container* 20 *feet* dan *container* 40 *feet* tidak bisa diletakkan bersamaan pada *bay* ganjil dan *bay* genap yang berurutan.

(10) dan (11) *Container* 20 *feet* tidak bisa diletakkan di atas *container* 40 *feet*.

(12) Total berat tumpukan *container* 20 *feet* pada satu *tier* tidak boleh melebihi batas MT.

(13) Total berat tumpukan *container* 40 *feet* pada satu *tier* tidak boleh melebihi batas MF.

(14) *Container* yang lebih berat tidak bisa diletakkan di atas *container* yang lebih ringan. Sesuai dengan aturan penataan berdasarkan berat, yaitu *container* yang lebih berat berada di bawah *container* yang lebih ringan.

(15) *Container* dengan tujuan awal diletakkan di atas *container* dengan tujuan akhir. Sesuai dengan aturan penataan berdasarkan tujuan, yaitu *container*

tujuan akhir yang di *unloading* paling akhir diletakkan di bawah *container* yang tujuan awal yang di *unloading* paling awal.

(16) Selisih total berat *container* pada bagian *anterior* dengan bagian *posterior* tidak boleh melebihi toleransi Q2

(17) Selisih total berat *container* pada sisi *left* dengan sisi *right* tidak boleh melebihi toleransi Q1 Sesuai dengan aturan penataan *container* berdasarkan keseimbangan kapal, yaitu perbedaan berat total *container* antara bagian sumbu kapal ke depan dan bagian sumbu kapal ke belakang tidak melebihi batas yang ditentukan.

(18) Variabel keputusan merupakan bilangan biner (0,1).

2.3 Algoritma *Particle Swarm Optimization* (PSO)

PSO adalah algoritma yang diusulkan oleh Kennedy dan Eberhart (1995) yang dibangun berdasarkan perilakau sebuah kawanan burung atau ikan. Algoritma PSO meniru perilaku sosial organisme ini, perilaku sosial terdiri dari tindakan individu dan pengaruh dari individu-individu lain dalam suatu kelompok. Kata partikel menunjukkan, misalnya, seekor burung dalam kawanan burung. Setiap individu atau partikel berperilaku dengan cara menggunakan kecerdasannya (*intelligence*) sendiri dan juga dipengaruhi kelompok kolektifnya. Dengan demikian, jika suatu partikel atau seekor burung menemukan jalan yang tepat atau pendek menuju sumber makanan, sisa kelompok yang lain juga akan segera mengikuti jalan tersebut meskipun lokasi mereka jauh dari kelompok tersebut.

Dalam *particle swarm optimization* (PSO), kawanan diasumsikan mempunyai ukuran tertentu dengan setiap partikel posisi awalnya terletak disuatu lokasi yang acak dalam ruang multidimensi. Setiap partikel diasumsikan meiliki dua karakteristik yaitu posisi dan kecepatan. Setiap partikel bergerak dalam ruang tertentu dan mengingat posisi terbaik yang pernah dilalui atau ditemukan terhadap sumber makanan atau nilai fungsi objektif. Setiap partikel menyampaikan informasi atau posisi terbaiknya kepada partikel yang lain dan menyesuaikan posisi dan kecepatan masing-masing berdasarkan informasi yang diterima mengenai posisi tersebut.

Prosedur dasar PSO (Santosa & Willy, 2011).

1. Lakukan inisialisasi

   Dilakukan inisialisasi seperti menentukan jumlah partikel, kecepatan partikel, jumlah iterasi.
2. Bangkitkan populasi

   Membangkitkan populasi awal, biasanya dibangkitkan secara random.
3. Tentukan kecepatan

   Kecepatan awal biasanya diasumsikan sama dengan 0. Semua partikel bergerak menuju titik optimal dengan suatu kecepatan tertentu.
4. Evaluasi fungsi tujuan dari masing-masing partikel
5. Temukan fungsi tujuan terbaik diantara partikel sekarang dengan partikel sebelumnya, kemudian tetapkan sebagai P_{best}. Untuk iterasi pertama P_{best} sama dengan partikel awal untuk semua partikel
6. Temukan fungsi tujuan terbaik diantara semua partikel kemudian tetapkan sebagai G_{best}
7. *Update* populasi

   Update populasi untuk masing-masing partikel. Model dasar yang dibangun oleh Kennedy dan Eberhart (1995) yang mengGambarkan mekanisme *updating* status partikel

$$V_i(t) = V_i(t-1) + c_1 r_1 [X_i^L - X_i(t-1)] + c_2 r_2 [X^G - X_i(t-1)] \quad (2.19)$$

$$X_i(t) = V_i(t) + X_i(t-1) \quad (2.20)$$

   Kecepatan partikel dalam PSO sebelumnya di *update* terlalu cepat dan nilai *minimum* fungsi tujuan yang dicari sering terlewati, sehingga Shi dan Eberhart (1998) melakukan modifikasi dengan menambahkan inersia θ untuk mengurangi kecepatan pada formula *update* kecepatan. Nilai inersia bervariasi secara linear dalam rentang 0.9 hingga 0.4. Nilai bobot inersia yang tinggi menambah porsi pencaria *global* (*global exploration*), sedangkan nilai yang rendah lebih menentukan pencarian *local* (*local searc*). Secara matematis perbaikan ini bisa dituliskan

$$V_i(t) = \theta V_i(t-1) + c_1 r_1[X_i^L - X_i(t-1)] + c_2 r_2[X^G - X_i(t-1)] \quad (2.21)$$

$$\theta(i) = \theta_{max} - (\frac{\theta_{max} - \theta_{min}}{i_{max}})i \quad (2.22)$$

Keterangan:

X = Posisi partikel

V = Kecepatan partikel

i = Indeks partikel

t = Iterasi ke-t

X_i^L = Mempresentasikan *locar best* dari partikel

X^G = Memprsentasikan *global best* dari seluruh kawanan

c_1 , c_2 = Konstanta

r_1 , r_2 = Bilangan random yang bernilai antara 0 sampai 1

8. Evaluasi fungsi tujuan dari masing-masing *update* pupolasi
9. Jadikan *update* populasi sebagai populasi awal, kemudian ulangi langkah 4 sampai langkah 9 hingga kriteria penghentian terpenuhi

2.4 Posisi Penelitian

Penelititan tentang *Container Stowage Problem* (CSP) telah banyak dilakukan dengan berbagai macam metoda dan algoritma, seperti *Genetic Algorithm* yang dilakukan oleh Dubrovsky et al (2002) menggunakan *compact encoding* untuk memperkecil pencarian, memberikan hasil yang memuaskan tapi jumlah iterasi sangat banyak. Kemudian Imai et al, (2006) melakukan penelitian dengan dengan algoritma yang sama dan menambahkan *tournament* pada *Genetic Algorithm* untuk solusi yang lebih baik, tapi waktu komputasi menjadi lebih lama. Kemudian Martins et al (2009) juga melakukan penelitian dengan *Genetic Algorithm* dengan memperhatikan keseimbangan *transverse* dan *longitudinal*, solusi baik tapi kasus yang digunakan bukan kasus yang kompleks dan besar. Kemudian Zhang et al, (2015) menggukan *A Bi-Level Genitic Algorithm*, model didasarkan pada *Mixet Integer Programming* untuk dua permasalahan yaitu masalah *quay crane operating time* adalah untuk mengurangi dan mengoptimalkan

bongkar muat, kemudian mengoptimalkan *stowage plan*. *A bi-level genitic algorithm* diusulkan untuk memecahkan model ini, dari percobaan numerik menunjukkan model dan algoritma lebih efektif dan tidak melebihi batas bawah, dan lebih baik jika dibandingkan dengan *Johnson's rule based heuristic algorithm method* (JHA) serta dengan metoda yang umum.

Selain *Genitic Algorithm* beberapa peneliti juga mencoba dengan algoritma metoda yang lain, seperti metoda *Heuristics* yang dilakukan oleh Ambrosino et al (2004) dengan menggunakan evaluasi eksak model 0-1 *Linear Programming*. Praktis untuk kasus sederhana, tapi tidak praktis untuk kasus besar. Kemudian Fan et al (2010) dengan metoda *heuristic* dibangun dengan *Block Selection* yang terdiri dari dua *Stage Block Selection*, yaitu *Block Rangking* yaitu dilakukan perengkingan pada *Port Of Destination* (POD) dan *Block Allocation*. Kemudian Monaco et al (2014) juga melakukan penelitian dengan *Hexuristics* yaitu melakukan *management* dengan tujuan meminimasi biaya yang berhubungan dengan *yard and transport operation*, mengusulkan *Binary Integer Programming* dan *Two-Step Heuristics Algorith*, langkah pertama adalah (CH) *Constructs an Initial Feasible Solution* dan langkah yang kedua (IH) *Search For Better Feasible Solution*. Kedua langkah tersebut didasarkan pada *Tabu Seach* yang merupakan *heuristics* pencarian lokal berulang dengan mekanisme memori. *Container* diklasifikasikan berdasarkan atribut, seperti ukuran (*standard, 45-fouter, hight cube, over-sized*), berat (*light, medium, heavy*), tipe (*reefer, open- top*), beban (bahaya, mudah rusak), dan *port of destination* (POD). Komputasi menunjukkan efisiensi dan efektifitas. Dan juga Ambrosino et al (2015) melakukan penelititan menggunakan *Mixet Integer Programming* dibagi berdasarkan tipe kontainer, dan tiap-tiap tipe dibagi tiga kelas yaitu *light, medium, heavy clas,* tidak menemukan solusi yang layak dalam waktu satu jam. Kemudian (Ding & Chou, 2015) dengan mengembangkan *Heuristics Algorithm* didapatkan hasil yang baik dibandingkan dengan *suspensory heuristics procedure (SH algorithm)*, tetapi hanya memuat *stowage planning* berdasarkan informasi pemuatan *port* saat ini saja, sedangkan algoritma SH memuat informasi dari semua *port.*

Selain *Genitic Algorithm* dan metoda *Heuristik* banyak algoritma dan metoda yang lain digunakan dalam penelititan *Container Stowage Problem* (CSP),

seperti yang dilakukan oleh Ambrosino et al (2010) dengan *Ant Colony Optimization*, cocok untuk kasus yang sangat besar, sedangkan untuk kasus yang sedang lebih baik menggukanan *Tabu Search*. Kemudian Wilson et al (2001) melakuakn penelitian dengan algoritma *Branch and Bound, Tabu Seach*, dibagi dalam dua fase, *Strategic planning process* dengan *Branch and Bound* dan *tactical planning process* dengan *Tabu Seach*, solusi dapat tercapai tapi membutuhkan waktu cukup lama. *Contraint Programming and Integer Programming* (Delgado et al, 2012) dengan dua pendekatan yaitu *Constraint Programming* dan *Integer Programming*, keduanya menghasilkan hasil yang bagus. Kemudian Lehnfeld & Knust (2014) melakukan *Survey Method and Classification* dilakukan pengedentifikasian pada kelas yaitu *Loading, Unloading, Premarshalling,* dan *Combained Problem*, kesimpulannya banyak permasalahan yang diselesaikan dengan cara eksak atau *heuristic methods* pada kelas *Loading* dan *Unloading.* Kemudian *Mixet Integer Linear Programming* (Wang, Liu, & Meng, 2015) diaplikasikan pada *shiping network of global* yaitu permasalahan pengiriman *container* dengan kapal. Model yang dikembangkan diterapkan untuk jaringan pengiriman kapal Asia-Eropa-Oceania dengan total 46 port dan 11 rute kapal. Hasil menunjukkan bahwa masalah dapat diselesaikan secara efisien dan jaringan dioptimalkan mengurangi total biaya. *Hybrid Metaheuristic and Local Searc Heuristics* (Araujo et al, 2016) menggunakan *Clustering Search* merupakan metoda *Hybrid* dari *Metaheuristics* dan *Heuristics*.Dibagi empat konsep, *Search Metaheuristics* (SM) untuk menjelajahi ruang pencarian dengan memanipulasi beberapa solusi, *Iterative Clustering* (IC) untuk mengumpulkan solusi yang sama kedalam kelompok, *Analyzer Module* (AM) untuk memeriksa setiap klaster, dan *Local Searcher* (LS) yaitu pencarian internal. PCS lebih unggul dalam solusi untuk *mono-objective method*, untuk *pareto fronts* unggul dalam semua kasus yaitu waktu dan solusi lebih baik. Kemudian Putamawa & Santosa (2011) melakukan penelitian dengan *Bee Swarm Optimization* (BSO) dengan melakukan modifikasi yaitu menambahkan langkah pengurutan berat tumpukan *container*, mendapatkan hasil yang lebih besar dari penelitian sepelumnya.

Dari penelitan terdahulu mengenai *Container Stowage Problem* (CSP) belum pernah dilakukan dengan menggunakan *Particle Swarm Optimization* (PSO)

dengan mempertimbangkan kelima faktor yaitu total berat *container*, total berat satu tumpukan *container*, keseimbangan kapal, *container* yang lebih berat dibawah *container* yang lebih ringan, dan tujuan *container*. Sehingga dalam penelititan ini melakukan penelititan pada kasus *Container Stowage Problem* (CSP) dengan menggunakan *Particle Swarm Optimization* (PSO).

Beberapa penelitian yang menjadi acuan karena memiliki permasalahan yang sejenis seperti pada Tabel 2.1.

Tabel 2.1. Tabel Penelitian Sebelumnya

NO	PENELITI	JUDUL	METODE	REVIEW
1	Dubrovsky et al (2002)	*A Genetic Algorithm with a Compact Solution Encoding for the Container Ship*	*Genitic Algorithm*	Menggunakan *compact encoding*
2	Imai et al (2002)	*Multi-objective simultaneous stowage and load planning for a container ship with container rehandle in yard stacks*	*Genitic Algorithm*	Menambahkan *tournament* pada *Genetic Algorithm*
3	Ambrosino et al (2004)	*Stowing a containership : the master bay plan problem*	*Heuristics*	Menggunakan evaluasi eksak model 0-1 *Linear Programming*
4	Martins et al (2009)	*Container Stowage Problem Solution for Short Sea Shipping*	*Genitic Algorithm*	Memperhatikan keseimbangan *transverse* dan *longitudinal*. *Genetic Algorithm*
5	Ambrosino et al (2010)	*Comparison of Diffirent Heuristics for the Master Bay Plan Problem*	*Ant Colony Optimization*	*Ant Colony Optimization*
6	Fan et al (2010)	*Stowage Planning of Large Containership with Trade off between Crane Work-load Balance and Ship Stability*	*Heuristics*	*Heuristic* dibangun dengan *Block Selection* yang terdiri dari dua *Stage block selection* , yaitu *Port Of Destination* (POD) dan *Block Allocation*
7	Wilson et al (2011)	*Container Stowage Pre-planning: Using Search to Generate Solution, A Case Study*	*Branch and Bound, Tabu Seach*	Dibagi dalam dua fase, *Strategic planning process* dengan *Branch and Bound* dan *tactical planning process* dengan *Tabu Seach*.
8	Putamawa & Santosa (2011)	Pengembangan algoritma *bee swarm optimization* untuk penyelesaian *container*	*Bee Swarm Optimization*	Menambahkan langkah pengurutan berat tumpukan *container*
9	Delgado et al (2012)	*A Constrain Programming Model for Fast Optimal Stowage of Container Vessel Bays*	*Contraint Programming and Integer*	Menggunakan dua pendekatan
10	Lehnfeld dan Knust (2014)	*Loading, unloading and premarshalling of stacks in storage areas: Survey and classification*	*Survey Method and Classification*	Dilakukan pengedentifikasian pada kelas yaitu *Loading, Unloading, Premarshalling,* dan *Combained Proble*
11	Monaco dan Sammarra dan Sorrentino	*The terminal-oriented ship stowage planning problem*	*Heuristics*	Mengusulkan *Binary Integer Programming* dan *Two-Step Heuristics Algorith*

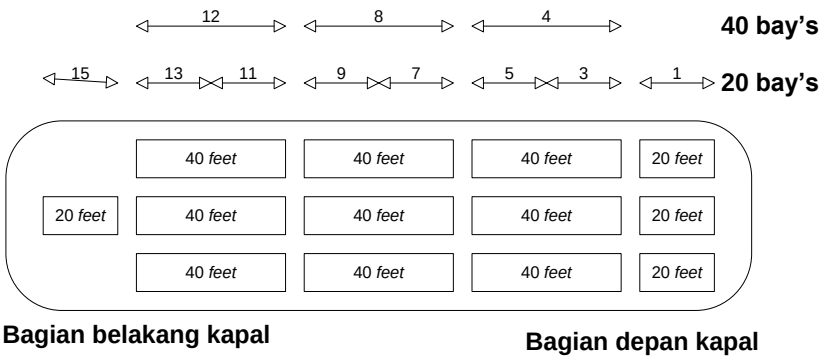
12	Ambrosino et al (2015)	*A MIP Heuristic for Multi Port Stowage Planning*	*Heuristics*	Menggunakan *Mixet Integer Programming*. Membagi berdasarkan tipe kontainer, dan tiap-tiap tipe dibagi tiga kelas yaitu *light, medium, heavy clas.*
13	Ding dan Chou (2015)	*Stowage Planning for Container Ships: A Heuristics Algorithm to Reduce The Number of Shifts*	*Heuristics*	*Heuristics algorithm* dibandingkan dengan *suspensory heuristics procedure (SH algorithm)*, tetapi hanya memuat *stowage planning* berdasarkan informasi pemuatan port saat ini saja, sedangkan algoritma SH memuat informasi dari semua port
14	Zhang et al (2015)	*Optimization for two-stage double-cycle operations in container terminals*	A Bi-Level Genitic Algorithm	Model didasarkan pada *Mixet Integer Programming* untuk dua permasalahan yaitu *quay crane operating time* dan s*towage plan . A bi-level genitic algorithm* diusulkan untuk memecahkan model ini.
15	Wang dan Liu dan Meng (2015)	*Segment-based alteration for container liner shipping network design*	*Mixet Integer Linear Programming*	*Mixet Integer Linear Programming* (MILP) diaplikasikan pada *shiping network of global*
16	Arauju et al (2016)	*Pareto Clustering Search Applied for 3D Container Ship Loading Plan Problem*	*Hybrid Metaheuristic and Local Searc Heuristics*	Menggunakan *Clustering Search* merupakan metode *Hybrid* dari *Metaheuristics* dan *Heuristics*. Dibagi empat konsep, *Search Metaheuristics* (SM), *Iterative Clustering* (IC), *Analyzer Module* (AM), dan *Local Searcher* (LS).
17	Matsaini & Santosa (2016)	*Menyelesaikan Container Stowage Problem (CSP) menggunakan Algorithm Particle Swarm Optimization (PSO)*	*Particle Swarm Optimization*	Menambahkan langkah penyesuaian kontainer berdasarkan *tier* , kemudian menambahkan langkah pengurutan kontainer berdasarkan berat satu tumpukan kontainer, dan menambahkan pengurutan kontainer berdasarkan tujuan kontainer

BAB III
METODOLOGI PENELITIAN

Metodologi penelitian digunakan sebagai pedoman agar penelitian dapat berjalan secara terstruktur sesuai kerangka penelitian, dan menguraikan metodologi penelitian yang digunakan serta tahapan pengembangan algoritma untuk menyelesaikan permasalahan yang dihadapi.

3.1 Metodologi Penelitian

Container Stowage Problem merupakan hal-hal yang berkaitan dengan penataan *container* kedalam kapal, penataan ini yang selalu dihadapai sehari-hari oleh masing-masing *terminal management* (Ambrosino et al., 2010). Beberapa fungsi kendala yang dihadapi dalam penataan *container* sehingga masalah penataan menjadi rumit dan susah dipecahkan dengan metoda eksak, diantaranya yaitu *container* yang lebih berat harus berada dibawah *container* yang lebih ringan agar tidak terjadi kerusakan pada *container* akibat tidak kuat menaham beban diatasnya, *container* 20 *feet* tidak bisa diletakkan bersamaan dengan *container* 40 *feet* pada *bay* yang sama, penomoran *bay* pada kapal dari depan kebelakan kapal seperti pada Gambar 3.1.



Gambar 3.1 Penataan *Container* Berdasarkan Ukuran *Container* pada *Bay*

Dalam satu tumpukan *container* tidak melebihi batas maksimum berat satu tumpukan sehingga tidak terjadi goyah pada tumpukan, berat *container* pada kapal harus memenuhi batas berat maksimum keseimbangan kapal agar tidak terjadi kecelakan pada saat berlayar, dan *container* dengan tujuan pertama diletakkan diatas *container* dengan tujuan akhir sehingga tidak terjadi *shifting* dan waktu *unloading* minimum.

Dengan adanya beberapa kendala permasalahan penataan *container* menjadi sulit untuk dipecahkan dengan perhitungan eksak, untuk memecahkan permasalahan yang rumit akan lebih tepat menggunakan algoritma metaheuristik, beberapa algoritma metaheuristik telah diaplikasikan pada kasus ini seperi yang telah dilakukan oleh Putamawa dan Santosa (2011) dengan menggunakan *Bee Swarm Optimization,* algoritma ini terinspirasi dari perilaku sekumpulan lebah madu dalam mencari makanannya, kemudian diaplikasikan sebagai komputasi dalam menyelesaikan penataan *container* kedalam kapal, algoritma ini berhasil diaplikasikan dengan dilakukannya suatu modifikasi pada algoritma tetapi belum menghasilkan hasil yang bagus, maka dari itu algoritma yang lain diusulkan dalam penelitian ini yaitu *Particle Swarm Optimization* (PSO).

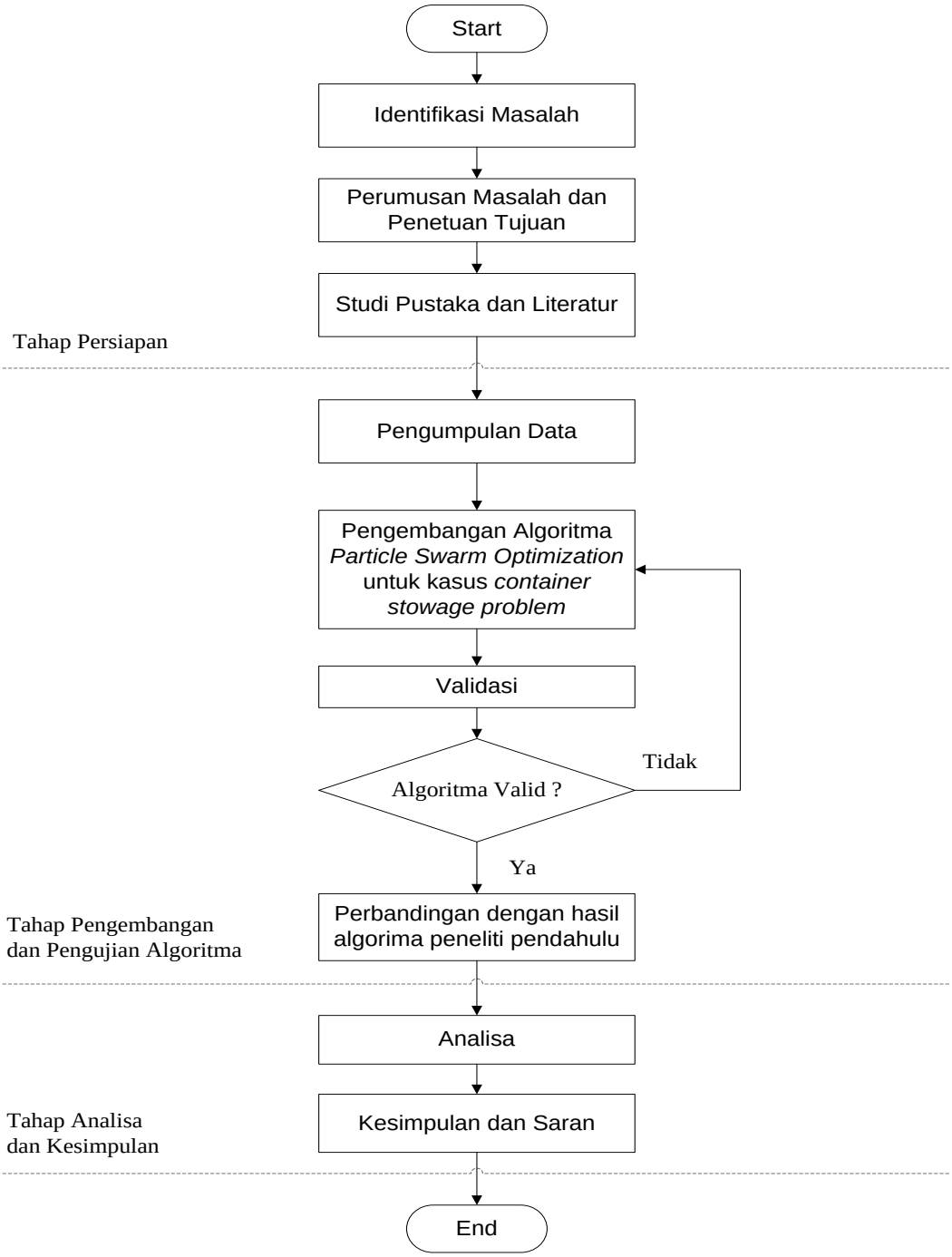
Algoritma *Particle Swarm Optimization* meniru perilaku sosial suatu kawanan burung atau ikan dalam mencari makanan, perilaku sosial terdiri dari tindakan individu dan pengaruh dari individu-individu lain dalam suatu kelompok. Kata partikel menunjukkan seekor burung dalam kawanan burung. Setiap partikel berperilaku dengan cara menggunakan kecerdasannya (*intelligence*) sendiri dan juga dipengaruhi kelompok kolektifnya. Dengan demikian, jika suatu partikel menemukan jalan yang tepat atau pendek menuju sumber makanan, sisa kelompok yang lain juga akan segera mengikuti jalan tersebut meskipun lokasi mereka jauh dari kelompok tersebut.

Penerapan algoritma PSO pada kasus penataan *container* perlu adanya suatu pengembangan untuk disesuaikan dengan persasalahan yang dihadapi sehingga bisa diaplikasikan pada permasalahan. Hasil dari pengembangan perlu dilakukan validasi apakah pengembangan algoritma telah mampu menyelesaikan masalah, validasi dilakukan dengan membandingkan hasil perhitungan numerik dengan hasil algoritma pada kasus yang sederhana, jika menghasilkan hasil yang sama maka

algoritma dikatakan valid, karena hasil perhitungan numerik pasti menghasilkan hasil yang optimal, tetapi hanya bisa dihitung pada kasus yang sangat sederhana. Kemudian penelitian bisa dilanjutkan ketahap selanjutnya. Data yang digunakan adalah data sekunder dari penelitian sebelumnya yang telah dilakukan Putamawa dan Santosa (2011) dan Ambrosino et al (2004, 2010). Hasil dari pengembangan algoritma *particle swarm optimization* (PSO) kemudian dibandingkan dengan hasil dari algoritma lain dari penelititan sebelumnya.

Tahap analisis dan kesimpulan adalah dilakukan analisis dari hasil perbandingan algoritma *particle swarm optimization* (PSO) dengan algoritma lain dari penelitian sebelumnya. Kemudian dari analisis yang telah dilakukan dapat diambil suatu kesimpulan.

Metodologi penelitian yang telah dijelaskan sebelumnya dapat dilihat pada Gambar 3.2.



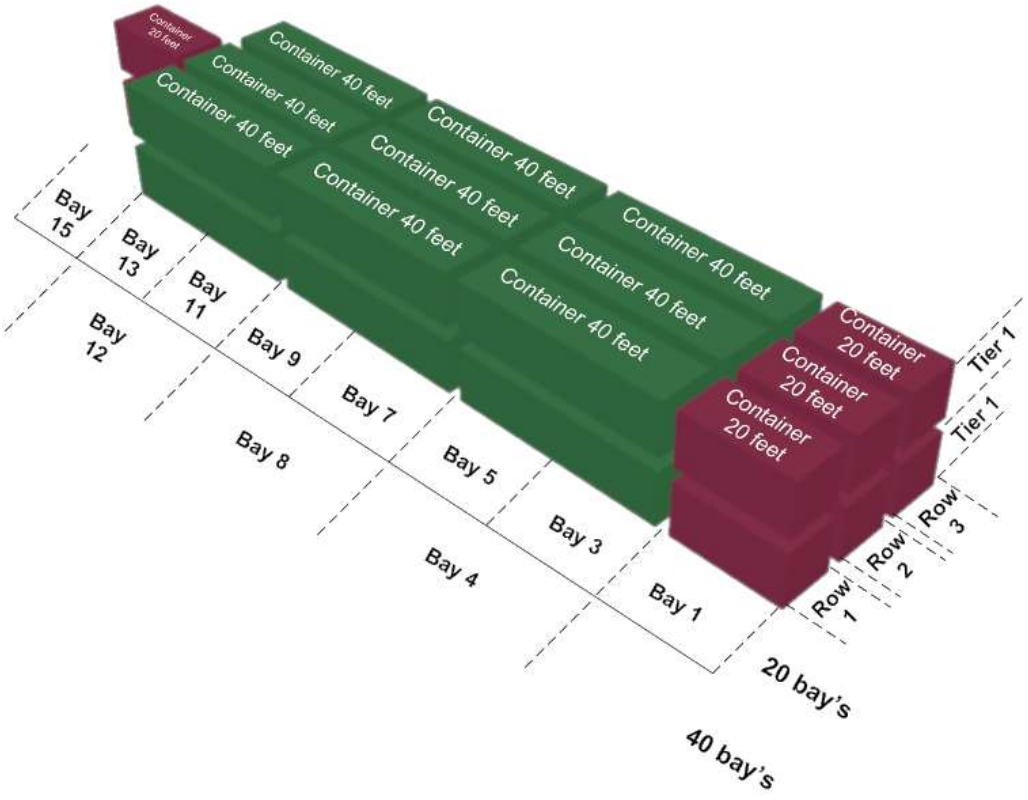
Gambar 3.2. *Flowchat* Metodologi Penelitian

3.2 Pengembangan algoritma

Untuk menerapkan *Particle Swarm Optimization* (PSO) untuk kasus *Container Stowage Problem* (CSP) perlu adanya penyesuaian atau modifikasi algoritma untuk disesuaikan terhadap permasalahan, modifikasi tersebut adalah sebagai berikut:

1. Pengumpulan data

   Mengumpulkan data tentang permasalahn CSP yang digunakan yaitu ukuran, berat, tujuan, waktu *unloading*, dan jumlah *bay (bay 1, bay 2, bay 3….)*, *row (row1, row2, row 3…)*, dan *tier (tier 1, tier 2, tier 3…)*. Contoh jumlah perhitungan jumlah *bay*, *row*, dan *tier* seperti pada Gambar 3.3 dengan 8 *bay*, 3 *row* dan 2 *tier*.



Gambar 3.3 Penomoran *Bay*, *Row*, dan *Tier*

2. Penentuan parameter

   Menentukan parameter awal seperti:

   - ✓ N adalah jumlah partikel
   - ✓ c_1 dan c_2 adalah suatu konstanta yang bernilai positif yang biasanya disebut dengan *learning factor*.
   - ✓ Kecepatan awal biasanya diasumsikan sama dengan nol
   - ✓ Solusi awal dibangkitkan dengan cara membangkitkan bilangan random
   - ✓ Menentukan bobot inersia θ. Biasanya digunakan nilai θ_{max} = 0.9 dan θ_{min} = 0.4
   - ✓ Q = Total kapasitas
   - ✓ Q1 = Toleransi keseimbangan *left-right*
   - ✓ Q2 = Toleransi keseimbangan *anterior-posterior*
   - ✓ MT = Batas berat satu tingkat *container 20 feet*
   - ✓ MF = Batas berat satu tingkat *container 40 feet*

3. Penentuan jumlah lokasi

   Menentukan jumlah lokasi untuk masing-masing *container* 20 *feet* dan 40 *feet* harus sama dengan total jumlah lokasi dalam kapal sesuai dengan persamaan (2). Untuk menetukan jumlah lokasi pada masing-masing *container* sebagai berikut:

   Container 20 = *bay* x *row* x *tier*
   Container 40 = (*bay*/2) x *row* x *tier*

4. Penentuan koordinat lokasi *container*

   Penamaan *bay*, *row*, dan *tier* pada kapal merupakan suatu koordinat lokasi yang bisa ditempati oleh *container*. Adapun cara pembangkitan kordinat lokasi adalah.

   a) Jumlah lokasi tiap *tier*
   b) Bangkitkan bilangan random sejumlah *bay* dan *row*.
   c) Urutkan bilangan random dari yang terkecil ke yang terbesar untuk mendapatkan indeksnya pada masing-masing *bay* dan *row*.
   d) Pada indeks *bay* dan *row* pilih angka pertama untuk dijadikan kordinat posisi *container* nomor ke-i, dimana i adalah nomor urut *container*.

e) Ulangi langkah b) sampai e), jika kordinat posisi yang baru dibangun ada yang sama dengan kordinat posisi sebelumnya yang telah dibangun, maka kordinat posisi yang baru dibangun tidak digunakan dan kembali kelangkah b).

f) Bangkitkan kordinat posisi sejumlah lokasi pada *tier* pertama. Dimana lokasi tiap *tier* selalu mengikuti lokasi *tier* yang dibawahnya.

5. Pembangkitan solusi awal sebanyak jumlah partikel (N)

   Pembangkitan solusi awal untuk masing-masing *container* 20 *feet* dan 40 *feet*. Solusi awal untuk masing-masing *container* 20 *feet* dan 40 *feet* dibangkitkan sesuai dengan jumlah lokasi pada *tier* ke-i dan sebanyak jumlah partikel. Solusi awal dibangkitkan melalui bilangan random (0-1) sejumlah populasi. Kemudian setiap partikel, urutkan bilangan random dari yang terkecil ke yang terbesar. Pengurutan ini akan menghasilkan solusi untuk setiap partikel.

6. Penyesuaian urutan dengan jumlah *container*

   Penyesuaian pada masing-masing *container* 20 *feet* dan 40 *feet*. Penyesuaian ini dilakukan karena jumlah lokasi lebih banyak dari jumlah *container*. Nomor pada urutan yang tidak ada dalam daftar nomor *container* (nomor yang lebih dari jumlah *container*) diubah menjadi 0 dan dianggap pada lokasi tersebut tidak ada *container* yang diletakkan. Dalam penyesuaian ini harus sesuai dengan persamaan (2.3) dan (2.4) yaitu satu *container* hanya ditempatkan pada satu lokasi, dan satu lokasi hanya bisa ditempati Satu *container*.

7. Penggabungan *container* 20 *feet* dan 40 *feet*

   Penggabungan *container* 20 *feet* dan 40 *feet* harus memenuhi persamaan (2.6) yaitu *container* 20 *feet* diletakkan pada *bay* ganjil dan persamaan (2.7) yaitu *container* 40 *feet* diletakkan pada *bay* genap atau dua *bay* ganjil. Serta harus sesuai dengan persamaan (2.8) dan (2.9) yaitu *container* 20 *feet* dan 40 *feet* tidak bisa diletakkan secara bersamaan pada *bay* ganjil dan *bay* genap yang berurutan. Jika tidak memenuhi persamaan, maka pilih salah satu jenis ukuran *container* untuk menyesuaikan terhadap jenis ukuran *container* lainnya. Pemilihan ukuran *container* yang akan disesuaikan

menggunakan skala perioritas secara random. Jenis ukuran *container* yang terpilih disesuaikan dengan jenis ukuran *container* yang tidak terpilih, penyesuaian dengan cara menggeser kearah depan atau kebelakang.

8. Penyesuaian menjadi posisi tumpukan

   Penyesuaian menjadi posisi tumpukan sesuai dengan persamaan (2.10) dan (2.11) yaitu *container* 20 *feet* tidak bisa diletakkan diatas *container* 40 *feet* atau sebaliknya. Dan juga harus memenuhi persamaan (2.12) dan (2.13) yaitu berat dalam satu tumpukan tidak melebihi batas berat maksimum. Posisi tumpukan pertama adalah *tier* 1, kemudian posisi tumpukan selanjutnya selalu mengikuti kordinat *bay* dan *row* pada *tier* sebelumnya. Ulangi langkah 4 sampai 8 hingga semua *container* pendapatkan lokasi penempatan.

9. Perhitungan nilai fungsi tujuan

   Nilai fungsi tujuan adalah penjumlahan dari total waktu *unloading* dan total dari nilai *penalty*

   a. Menghitung nilai total waktu *unloading*

      Menghitung waktu *unloading* berdasarkan matriks *time.* Waktu *unloading* tiap-tiap *container* berbeda-beda setiap lokasi. Jika disuatu lokasi tidak ada *container* maka waktu *unloading* sama dengan 0. Keseluruhan waktu *unloading container* dijumlahkan dan diperoleh total waktu *unloading* sesuai persamaan 1.

   b. Menghitung nilai *penalty*

      Perhitungan nilai *penalty* berdasarkan persamaan kendala yang dilanggar.

      - *Penalty* 1000000 ketika total berat *container* melebihi kapasitas kapal (Q) tidak sesuai dengan persamaan (2.5). Dan melebihi batas maksimal satu tumpukan *container* ukuran 20 (MT) dan *container* ukuran 40 (MF). *Penalty* 1000000 diberikan untuk total berat *conatainer* dan satu tumpukan *container* agar memperkecil peluang setiap solusi melanggar konstrain ini dan bahkan tidak mungkin dipilih karena diberikan angka yang sangat besar.

- *Penalty* 10000 ketika ada *container* yang lebih berat berada di atas *container* yang lebih ringan tidak sesuai dengan persamaan (2.14). Diberi angka *penalty* 10000 untuk memperkecil peluang setiap solusi melanggar konstrain, tetapi angka 10000 lebih kecil dari *penalty* yang diberikan untuk konstrain total berat container dan berat satu tumpukan container yang berarti masih ada kemungkinan solusi yang melanggar konstrain diterima tetapi kemungkinan itu sangat kecil.
- *Penalty* 100 ketika keseimbangan kapal melebihi batas yang ditentukan, tidak sesuai dengan persamaan (2.16) dan (2.17). Diberi angka *penalty* 100 untuk memperkecil peluang setiap solusi melanggar konstrain, namun pada konstain ini angka *penalty* lebih kecil dari angka *penalty* yang diberikan pada total berat *container*, total berat satu tumpukan *container*, dan *container* yang lebih berat berada di atas *container* yang lebih ringan, artinya kemungkinan terpilih solusi yang melanggar konstrain ini lebih besar.
- *Penalty* 1 ketika ada *container* tujuan awal berada di bawah *container* tujuan akhir tidak sesuai dengan persamaan (2.15). Diberi angka 1 dan lebih kecil dari konstrain yang lain karena besar kemungkinan konstrain ini akan selalu dilanggar, tetapi dengan pemberian angka 1 ini juga akan memunculkan solusi yang tidak melanggar konstrain ini.

10. *Update* populasi

    Temukan P_{best} untuk setiap partikel, P_{best} adalah nilai terbaik diantara partikel sekarang dan partikel sebelumnya. Untuk setiap partikel, P_{best} awal akan sama dengan nilai partikel awal. Kemudian temukan G_{best} dari semua partikel. G_{best} adalah nilai terbaik diantara semua partikel. P_{best} dan G_{best} adalah nilai bilangan random yang dibangkitkan, bukan solusinya. Kemudian lakukan *update* untuk kecepatan awal, misalnya partikel ke 1 bisa di *update* dengan Rumus (2.21) menjadi

$$V_{(2,1)} = \left[\theta_{max} - (\frac{\theta_{max}-\theta_{min}}{i_{max}})i\right] V_{(1,1)} + c_1 r_1[P_{best2,1} - X_{(1,1)}] +$$

$$c_2 r_2[G_{best} - X_{(1,1)}]$$

Kemudian untuk partikel 1 nilainya di *update* sesuai dengan Rumus (2.20) menjadi

$$X_{(2,1)} = V_{(2,1)} + X_{(1,1)}$$

Kemudian dilakukan pengecekan apakah partikel setelah di *update* memenuhi batas bawah dan batas atas. Jika melewati batas bawah atau batas atas maka ubah nilai setiap partikel pada nilai batas bawah atau batas atas.

11. Rubah posisi tumpukan
    - Rubah posisi tumpukan berdasarkan *tier*
    - Rubah posisi tumpukan berdasarkan tujuan
    - Rubah posisi tumpukan berdasarkan berat

    a. Menghitung nilai total waktu *unloading*

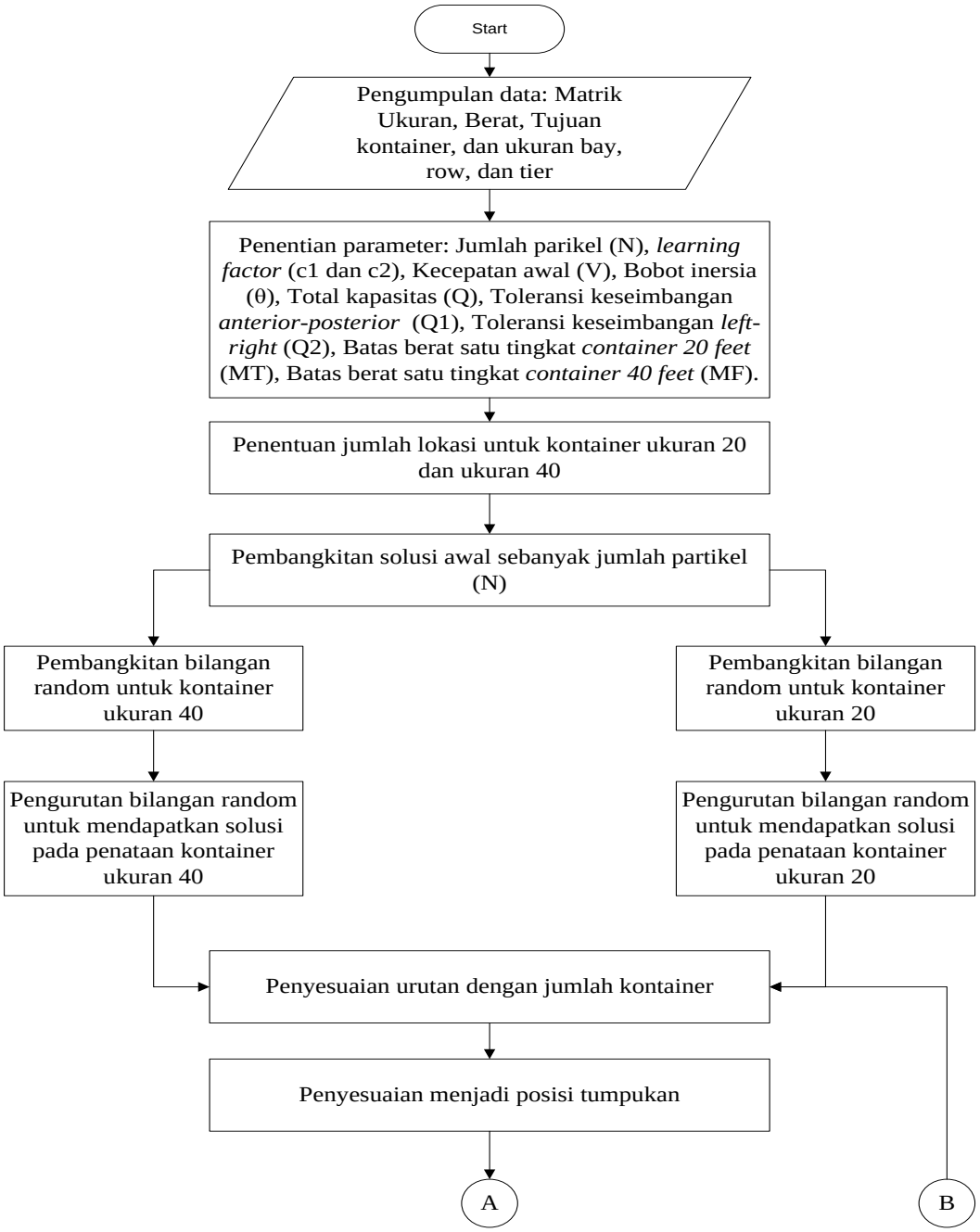
    b. Menghitung nilai total penalty

    Perhitungan nilai *penalty* sesuai dengan persamaan kendala yang dilanggar.
    - *Penalty* 1000000 ketika total berat *container* melebihi kapasitas kapal (Q) tidak sesuai dengan persamaan (2.5). Dan melebihi batas maksimal satu tumpukan *container* ukuran 20 (MT) dan *container* ukuran 40 (MF).
    - *Penalty* 100 ketika keseimbangan kapal melebihi batas yang ditentukan, tidak sesuai dengan persamaan (2.16) dan (2.17).

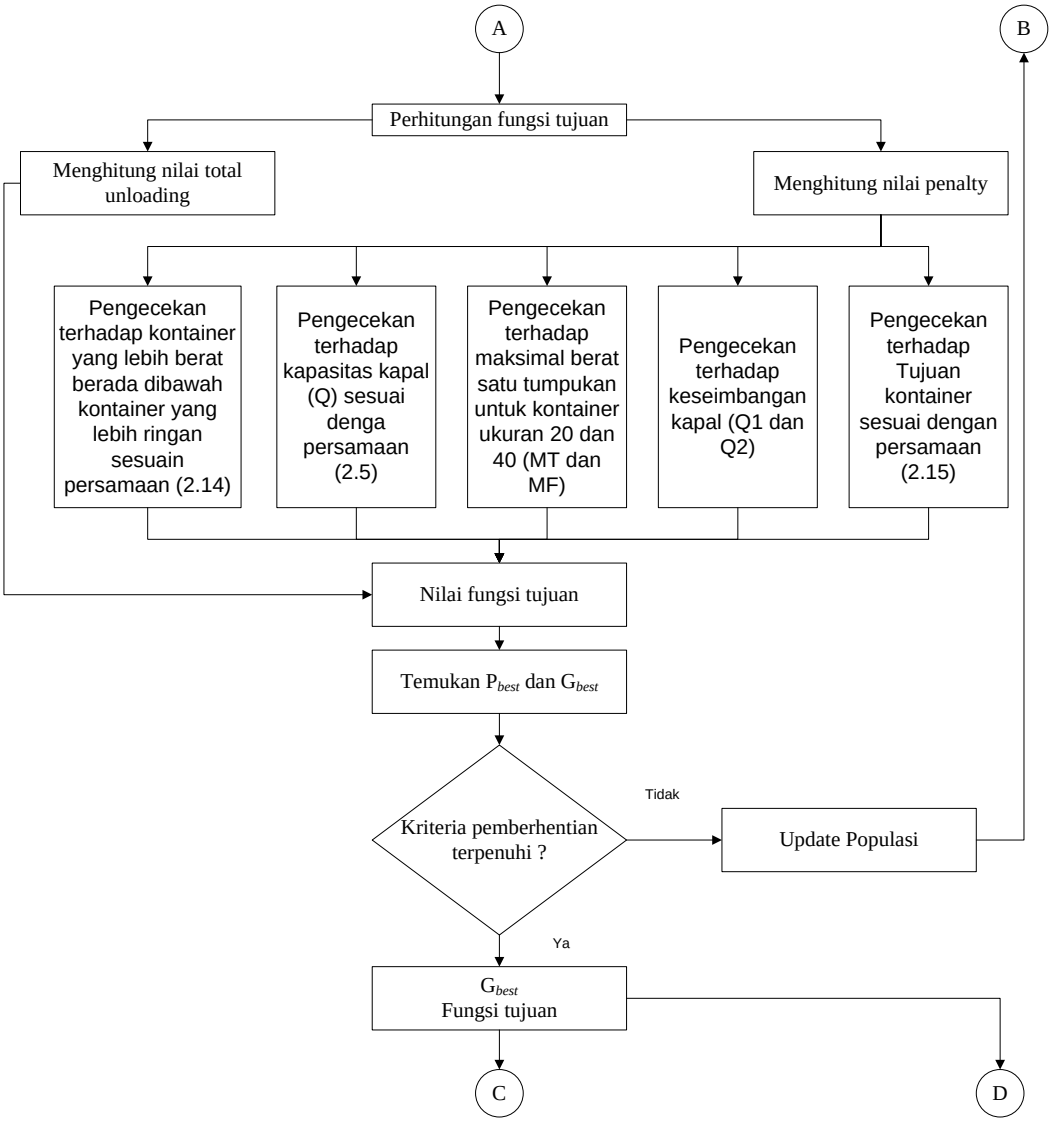
    Nilai fungsi tujuan pada langkah 11 dibandingkan dengan fungsi tujuan pada langkah 9 dan pilih nilai fungsi tujuan yang lebih kecil.

12. Ulangi langkah 5 sampai 11

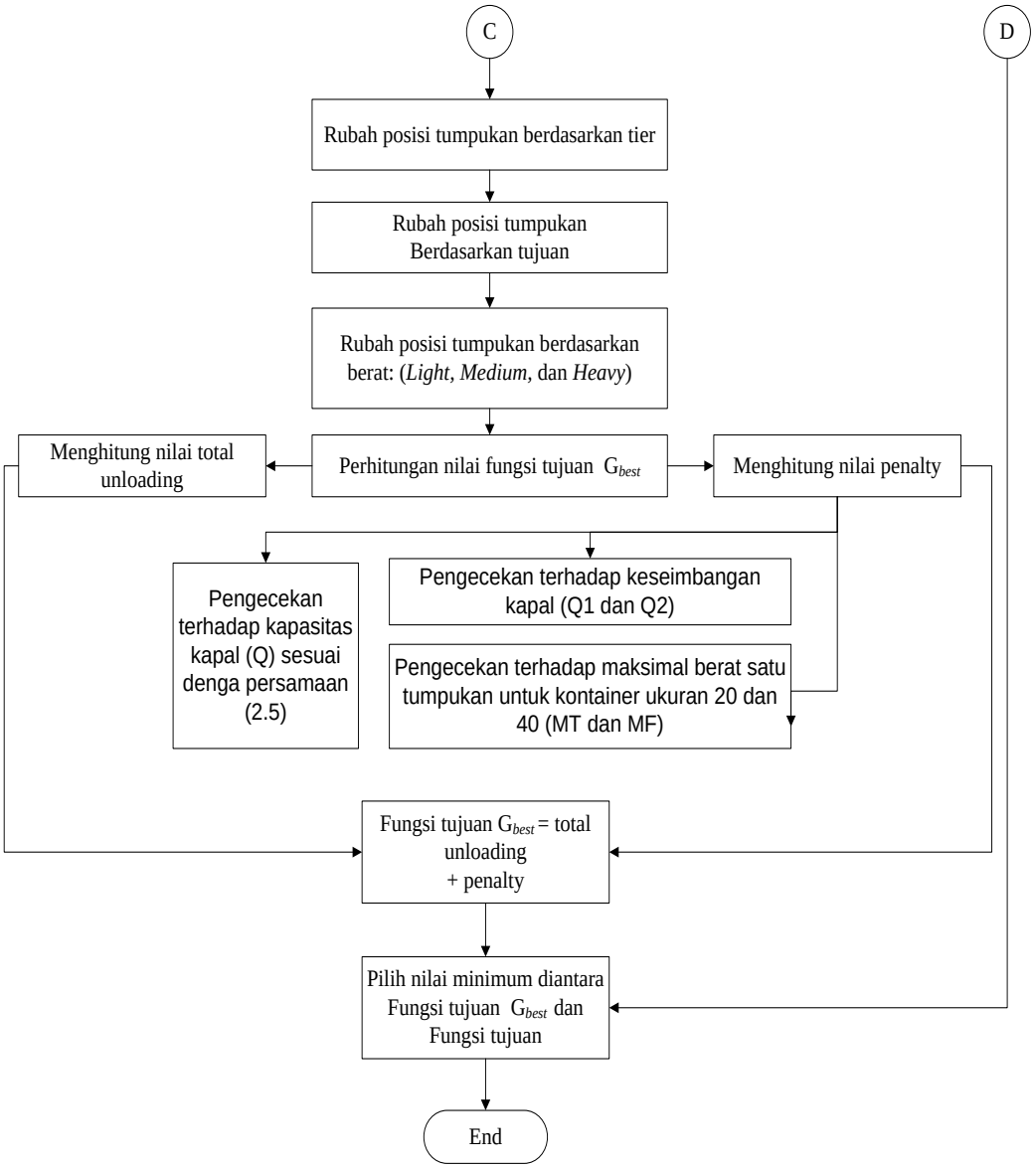
Dari langkah-langkah pengembangan algoritma yang telah dijelaskan sebelumnya dibuat *flowchart* seperti pada Gambar 3.4, Gambar 3.5, dan Gambar 3.6.



Gambar 3.4. *Flowchat* Pengembangan Algoritma *Particle Swarm Optimization*



Gambar 3.5. *Flowchat* Pengembangan Algoritma *Particle Swarm Optimization* (lanjutan)



Gambar 3.6. *Flowchat* Pengembangan Algoritma *Particle Swarm Optimization* (lanjutan)

3.3 Contoh Numerik

Pada contoh numerik ini ditunjukkan contoh penggunaan algoritma sekaligus proses validasi. Kasus yang digunakan adalah kasus yang sangat sederhana, yaitu penataan *container* pada lokasi yang berukuran 4 *bay*, 2 *row*, dan 1 *tier*, dengan data seperti pada Tabel 3.1 dan Tabel 3.2.

Tabel 3.1. Contoh Kasus Sederhana

Ukuran	Nomor	Berat	Tujuan
20 *feet*	1	10	1
	2	15	2
40 *feet*	1	25	1

Tabel 3.2. Waktu *Unloading*

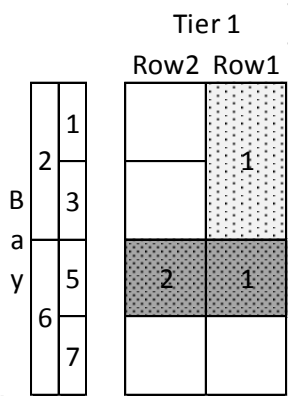
	*Row*1	*Row*2
Tier 1	120	126

Kasus ini akan diselesaikan dengan algoritma *particle swarm optimization* yang telah dikembangkan dan akan dibandingkan dengan teknik enumerasi. Jika hasilnya sama dengan teknik enumerasi, maka algoritma telah valid karena teknik enumerasi pasti menghasilkan solusi optimal.

3.3.1 Enumerasi

Enumerasi dilakukan dengan mencari semua kombinasi yang mungkin menjadi solusi. Untuk contoh kasus yang telah disebutkan sebelumnya terdapat 120 kemungkinan solusi. Setelah dilakukan perhitungan untuk semua kemungkinan

tersebut, diperoleh hasil terbaik seperti pada Gambar 3.7, dengan total fungsi tujuan = 366, penalti = 0, dan waktu *unloading* = 366



Gambar 3.7. Contoh Solusi Terbaik (dilihat dari atas)

3.3.2 Pengembangan Algoritma *Particle Swarm Optimization*

Contoh kasus sederhana diatas juga diselesaikan dengan pengembangan algoritma *particle swarm optimization* (PSO). Berikut langkah-langkah sesuai dengan langkah-langkah pengembangan algoritma yang telah dijelaskan sebelumnya.

1. Pengumpulan data

   Pengumpulan data sesuai dengan data contoh kasus sederhana.

2. Penentuan parameter

   N = 10

   c1 dan c2 = 1

   Kecepatan awal = 0

   θ_{max} = 0.9 dan θ_{min} = 0.4

   Q = 250 (Total kapasitas kapal)

   Q1 = 20 (Toleransi keseimbangan *left-right*)

   Q2 = 40 (Toleransi keseimbangan *anterior-posterior*)

   MT = 40 (Batas berat satu tingkat *container 20 feet*)

   MF = 80 (Batas berat satu tingkat *container 40 feet*)

3. Penentuan jumlah lokasi

Lokasi berukuran 4 *bay*, 2 *row*, dan 1 *tier*. Ketika *container* 20 *feet* diletakkan, maka *bay* diberi nomor ganjil dan disebut *bay* ganjil. Tapi, ketika *container* 40 *feet* diletakkan maka *bay* diberi nomor genap atau dua *bay* ganjil. Jumlah lokasi untuk masing-masing *container* sama dengan total jumlah lokasi dalam kapal sesuai dengan persamaan (2.2). Jumlah lokasi untuk masing-masing *container* seperti pada Gambar 3.8.

Container 20 = *bay* x *row* x *tier*

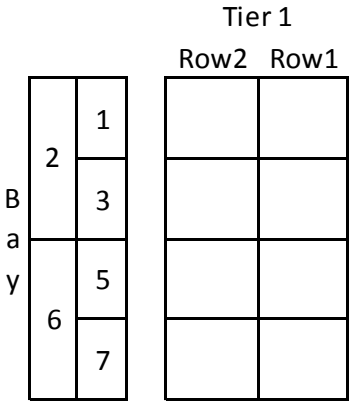
= 4 x 2 x 1

= 8 lokasi

Container 40 = (*bay*/2) x *row* x *tier*

= (4/2) x 2 x 1

= 4 lokasi



Gambar 3.8. Contoh Jumlah Lokasi (dilihat dari atas)

4. Penentuan koordinat lokasi *container*

Contoh koordinat posisi *bay*=4 (*bay 1*, *bay 2*, *bay 3*, *bay* 4) dan *row*=2 (*row 1*, *row 2*) pada *tier* 1

a) Jumlah lokasi pada *tier* 1 untuk *container* ukuran 20 = 8 lokasi, dan *container* ukuran 40 = 4 lokasi.
b) Bangkitkan bilangan random sejumlah *bay* dan *row*.

Bay = 0.1576 0.5469 0.9575 0.9649

Row = 0.9572 0.9706

c) Indeks dari hasil pengurutan bilangan random dari yang terkeicil ke yang terbesar

Bay = 4 1 2 3

Row = 2 1

d) Ambil angka pertama sebagai koordinat posisi pada *container* 1 = [4 2]

e) Ulangi langkah b) sampai e).

f) Bangkitkan kordinat posisi sejumlah *bay* dan *row*.

Bangkitkan bilangan random

Bay = 0.1419 0.4218 0.8003 0.9157

Row = 0.7922 0.9595

Koordinat posisi dari indeks [2 1]

Apakah indeks yang baru [2 1] sama dengan salah satu kolom pada matrik [4 2], jika sama maka indeks yang baru tidak digunakan dan ulangi langkah b) sampai e). Karena tidak ada yang sama, maka indeks yang baru dimasukkan kedamalam matriks sebelumnya sehingga menjadi $\begin{bmatrix} 4 & 2 \\ 2 & 1 \end{bmatrix}$

Kembali kelangkah b) hingga stoping kriteria terpenuhi.

Bay = 0.3171 0.6948 0.8235 0.9502

Row = 0.0344 0.4387

Koordinat posisi dari indeks [3 1]

Koordinat posisi menjadi $\begin{bmatrix} 4 & 2 \\ 2 & 1 \\ 3 & 1 \end{bmatrix}$

Bay = 0.1869 0.4898 0.7655 0.7952

Row = 0.4456 0.6463

Koordinat posisi dari indeks [3 1]

Koordinat posisi menjadi $\begin{bmatrix} 4 & 2 \\ 2 & 1 \\ 3 & 1 \end{bmatrix}$

Bay = 0.2760 0.6551 0.6797 0.7547

Row = 0.1190 0.1626

Koordinat posisi dari indeks [2 2]

Koordinat posisi menjadi $\begin{bmatrix} 4 & 2 \\ 2 & 1 \\ 3 & 1 \\ 2 & 2 \end{bmatrix}$

Ulangi langkah b) samapi e) sehingga baris pada matrik mencapai 8 seperti berikut.

Container 20 $= \begin{bmatrix} 4 & 2 \\ 2 & 1 \\ 3 & 1 \\ 2 & 2 \\ 4 & 1 \\ 1 & 2 \\ 1 & 1 \\ 3 & 2 \end{bmatrix}$

Container 40 $= \begin{bmatrix} 1 & 2 \\ 1 & 1 \\ 2 & 1 \\ 2 & 2 \end{bmatrix}$

Pada matrik *container* 40 untuk *bay*, angka 1 = 2, angka 2 = 6, sesuai dengan Gambar 3.7. Sehingga menjadi *Container* 40 $= \begin{bmatrix} 2 & 2 \\ 2 & 1 \\ 6 & 1 \\ 6 & 2 \end{bmatrix}$

5. Pembangkitan solusi awal sebanyak jumlah partikel (N)

Pembangkitan solusi awal dengan pembangkitan bilangan random, kemudian dari bilangan random tersebut diurutkan untuk menghasilkan solusi awal. Pembangkitan bilangan random dilakukan pada masing-masing ukuran *container* 20 *feet* dan 40 *feet.*

Bilangan random untuk ukuran *container* 20 *feet* sebagai berikut.

0.8147	0.1576	0.6557	0.7060	0.4387	0.2760	0.7513	0.8407
0.9058	0.9706	0.0357	0.0318	0.3816	0.6797	0.2551	0.2543
0.1270	0.9572	0.8491	0.2769	0.7655	0.6551	0.5060	0.8143
0.9134	0.4854	0.9340	0.0462	0.7952	0.1626	0.6991	0.2435
0.6324	0.8003	0.6787	0.0971	0.1869	0.1190	0.8909	0.9293
0.0975	0.1419	0.7577	0.8235	0.4898	0.4984	0.9593	0.3500
0.2785	0.4218	0.7431	0.6948	0.4456	0.9597	0.5472	0.1966
0.5469	0.9157	0.3922	0.3171	0.6463	0.3404	0.1386	0.2511
0.9575	0.7922	0.6555	0.9502	0.7094	0.5853	0.1493	0.6160
0.9649	0.9595	0.1712	0.0344	0.7547	0.2238	0.2575	0.4733

Kemudian bilangan random *container* 20 *feet* diurutkan seperti berikut.

0.1576	0.2760	0.4387	0.6557	0.7060	0.7513	0.8147	0.8407
0.0318	0.0357	0.2543	0.2551	0.3816	0.6797	0.9058	0.9706
0.1270	0.2769	0.5060	0.6551	0.7655	0.8143	0.8491	0.9572
0.0462	0.1626	0.2435	0.4854	0.6991	0.7952	0.9134	0.9340
0.0971	0.1190	0.1869	0.6324	0.6787	0.8003	0.8909	0.9293
0.0975	0.1419	0.3500	0.4898	0.4984	0.7577	0.8235	0.9593
0.1966	0.2785	0.4218	0.4456	0.5472	0.6948	0.7431	0.9597
0.1386	0.2511	0.3171	0.3404	0.3922	0.5469	0.6463	0.9157
0.1493	0.5853	0.6160	0.6555	0.7094	0.7922	0.9502	0.9575
0.0344	0.1712	0.2238	0.2575	0.4733	0.7547	0.9595	0.9649

Dari pengurutan bilangan random *container* ukuran 20 *feet* tersebut akan menghasilkan solusi awal untuk penataan *container* seperti berikut.

2	6	5	3	4	7	1	8
4	3	8	7	5	6	1	2
1	4	7	6	5	8	3	2
4	6	8	2	7	5	1	3
4	6	5	1	3	2	7	8
1	2	8	5	6	3	4	7
8	1	2	5	7	4	3	6
7	8	4	6	3	1	5	2
7	6	8	3	5	2	4	1
4	3	6	7	8	5	2	1

Pembangkitan bilangan random untuk ukuran *container* 40 *feet* sebagai berikut.

0.3517	0.0759	0.1622	0.4505
0.8308	0.0540	0.7943	0.0838
0.5853	0.5308	0.3112	0.2290
0.5497	0.7792	0.5285	0.9133
0.9172	0.9340	0.1656	0.1524
0.2858	0.1299	0.6020	0.8258
0.7572	0.5688	0.2630	0.5383
0.7537	0.4694	0.6541	0.9961
0.3804	0.0119	0.6892	0.0782
0.5678	0.3371	0.7482	0.4427

Kemudian bilangan random *container* 40 *feet* diurutkan seperti berikut.

0.0759	0.1622	0.3517	0.4505
0.0540	0.0838	0.7943	0.8308
0.2290	0.3112	0.5308	0.5853
0.5285	0.5497	0.7792	0.9133
0.1524	0.1656	0.9172	0.9340
0.1299	0.2858	0.6020	0.8258
0.2630	0.5383	0.5688	0.7572
0.4694	0.6541	0.7537	0.9961
0.0119	0.0782	0.3804	0.6892
0.3371	0.4427	0.5678	0.7482

Dari pengurutan bilangan random *container* ukuran 40 *feet* tersebut akan menghasilkan solusi awal untuk penataan *container* seperti berikut.

2	3	1	4
2	4	3	1
4	3	2	1
3	1	2	4
4	3	1	2
2	1	3	4
3	4	2	1
2	3	1	4
2	4	1	3
2	4	1	3

6. Penyesuaian urutan dengan jumlah *container*

Nomor pada urutan yang tidak ada dalam daftar nomor *container* (nomor yang lebih dari jumlah *container*) diubah menjadi 0 dan dianggap pada lokasi tersebut tidak ada *container* yang diletakkan.

Container 20

2	0	0	0	0	0	1	0
0	0	0	0	0	0	1	2
1	0	0	0	0	0	0	2
0	0	0	2	0	0	1	0
0	0	0	1	0	2	0	0
1	2	0	0	0	0	0	0
0	1	2	0	0	0	0	0
0	0	0	0	0	1	0	2
0	0	0	0	0	2	0	1
0	0	0	0	0	0	2	1

Kordinat pada *bay* dan *row container* 20 = $\begin{bmatrix} 4 & 2 \\ 2 & 1 \\ 3 & 1 \\ 2 & 2 \\ 4 & 1 \\ 1 & 2 \\ 1 & 1 \\ 3 & 2 \end{bmatrix}$

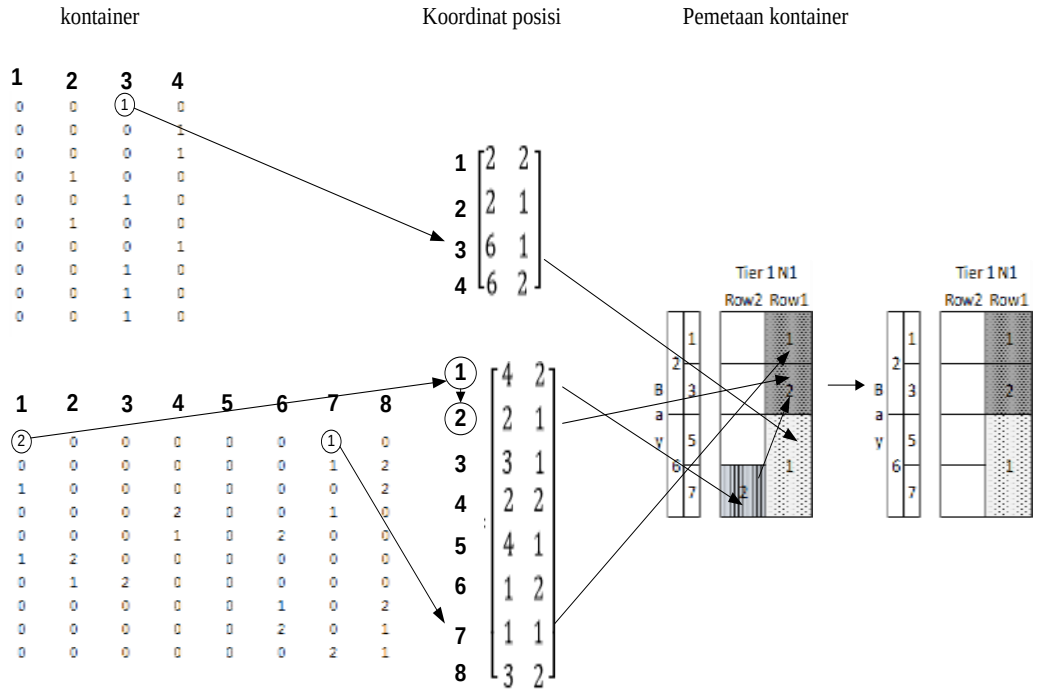
Container 40

0	0	1	0
0	0	0	1
0	0	0	1
0	1	0	0
0	0	1	0
0	1	0	0
0	0	0	1
0	0	1	0
0	0	1	0
0	0	1	0

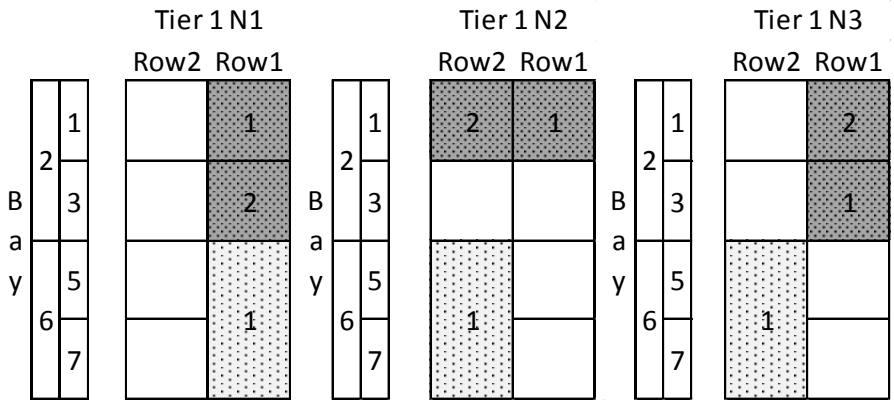
Kordinat pada *bay* dan *row container* 40 = $\begin{bmatrix} 2 & 2 \\ 2 & 1 \\ 6 & 1 \\ 6 & 2 \end{bmatrix}$

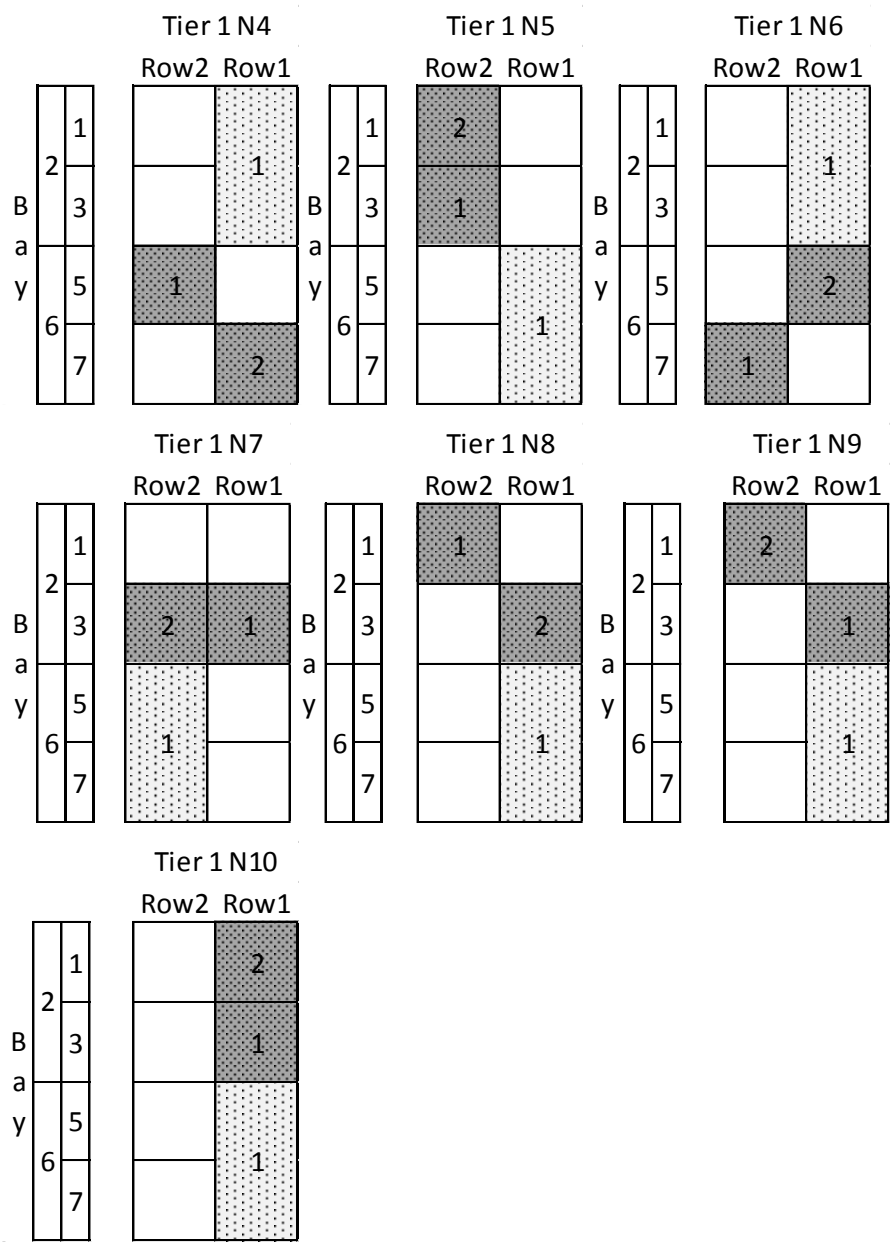
7. Penggabungan *container* 20 *feet* dan 40 *feet*

Penggabungan *container* 20 *feet* dan 40 *feet* harus memenuhi persamaan (2.6) dan persamaan (2.7). Serta harus sesuai dengan persamaan (2.8) dan (2.9). Jika tidak memenuhi persamaan, maka pilih salah satu jenis ukuran *container* untuk menyesuaikan terhadap jenis ukuran *container* lainnya. Pemilihan ukuran *container* yang akan disesuaikan menggunakan skala perioritas secara random. Jenis ukuran *container* yang terpilih disesuaikan dengan jenis ukuran *container* yang tidak terpilih, penyesuaian dengan cara menggeser kordinat pada matrik koordinat posisi. Misalnya *container* 20 terpilih secara random maka *container* 20 disesuaikan atau dilakukan pergeseran. Cara penggabungan *container* seperti pada Gambar 3.9 dan Gambar 3.10.



Gambar 3.9 Penggabungan *Container* 20 *Feet* dan 40 *Feet*





Gambar 3.10. Hasil Penataan *Container* pada Kasus Sederhana (dilihat dari atas)

8. Penyesuaian menjadi posisi tumpukan

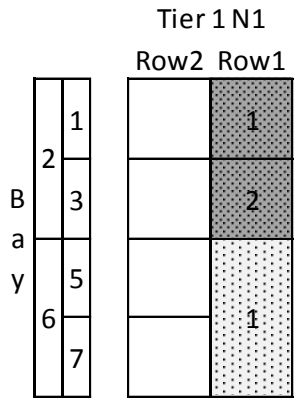
Penyesuaian menjadi posisi tumpukan sesuai dengan persamaan (2.10) dan (2.11) yaitu *container* 20 *feet* tidak bisa diletakkan diatas *container* 40 *feet* atau sebaliknya. Dan juga harus memenuhi persamaan (2.12) dan (2.13) yaitu berat

dalam satu tumpukan tidak melebihi batas berat maksimum. Namun karena hanya ada 1 *tier* maka persamaan tersebut langsung terpenuhi.

9. Perhitungan nilai fungsi tujuan

   a) Menghitung nilai total waktu *unloading*

      Total waktu *unloading* berdasarkan penataan *container* dengan waktu berdasarkan waktu *unloading*, seperti N1 pada Gambar 3.11.



Gambar 3.11. Hasil Penataan *Container* pada N1 (dilihat dari atas)

Container 1 ukuran 40 *feet* terletak pada *row* 1, *tier* 1 sebesar 120, *container* 1 ukuran 20 *feet* terletak pada *row* 1, *tier* 1 sebesar 120, dan *container* 2 ukuran 20 *feet* terletak pada *row* 1, *tier* 1 sebesar 120. Sehingga keseluruhan waktu *unloading* dijumlahkan sesuai dengan persamaan (2.1). Pada N1 total waktu *unloading* adalah 360. Begitu juga untuk N selanjutnya, sehingga diperoleh total waktu *unloading* seperti pada Tabel 3.3.

Tabel 3.3. Total Waktu *Unloading* pada Kasus Sederhana

N1= 360	N6= 366
N2= 372	N7= 372
N3= 366	N8= 366
N4= 366	N9= 366
N5= 372	N10= 360

b) Menghitung nilai *penalty*

Contoh perhitungan penalty pada N1.

- *Penalty* 1000000 ketika total berat *container* melebihi kapasitas kapal (Q) tidak sesuai dengan persamaan (2.5). Total berat N1 adalah 50 lebih kecil dari Q = 250, dan tidak melanggar batas maksimum satu tumpukan yaitu MT dan MF. Sehingga nilai *penalty* adalah 0
- *Penalty* 10000 ketika ada *container* yang lebih berat berada di atas *container* yang lebih ringan tidak sesuai dengan persamaan (2.14). Pada N1 tidak melanggar persamaan (2.14) sehingga nilai *penalty* adalah 0.
- *Penalty* 100 ketika keseimbangan kapal melebihi batas yang ditentukan, tidak sesuai dengan persamaan (2.16) dan (2.17). Pada persamaan (2.16) selisih total berat *container* pada bagian *anterior* dan *posterior* harus berada diantara –Q2 dan Q2. Pada persamaan (2.17) selisih total berat *container* pada sisi *left* dan *right* harus berada diantara –Q1 dan Q1. Persamaan (2.16) dan (2.17) secara matematis dapat ditulis seperti berikut.

$$-Q2 = -40 \leq (10+ 15) - 25 = 0 \leq Q2 = 40$$

$$-Q1 = -20 \leq (10 + 15+25) - 0 =50 \leq Q1 = 20$$

Dari kedua persamaan tersebut melanggar konstrain Q1, sehingga nilai *penalty* adalah 100

- *Penalty* 1 ketika ada *container* tujuan awal berada di bawah *container* tujuan akhir tidak sesuai dengan persamaan (2.15). Sehingga nilai *penalty* adalah 0.

  Sehingga total *penalty* pada N1 adalah 0 + 0 + 100 + 0 =100. Begitu juga untuk N selanjutnya, sehingga diperoleh seperti pada Tabel 3.4 dan Tabel 3.5.

Tabel 3.4. Total Penalti pada Kasus Sederhana

N1= 100	N6= 100
N2= 100	N7= 100
N3= 0	N8= 100
N4= 100	N9= 0
N5= 0	N10= 100

Tabel 3.5. Nilai Fungsi Tujuan pada Kasus Sederhana

N1= 360 + 100 = 460	N6= 366 + 100 = 466
N2= 372 + 100 = 472	N7= 372 + 100 = 472
N3= 366 + 0 = 366	N8= 366 + 100 = 466
N4= 366 + 100 = 466	N9= 366 + 0 = 366
N5= 372 + 0 =372	N10= 360 + 100 = 466

10. *Update* populasi

Temukan P_{best} untuk setiap partikel, P_{best} adalah nilai terbaik diantara partikel sekarang dan partikel sebelumnya. Untuk setiap partikel, P_{best} awal akan sama dengan nilai partikel awal. Kemudian temukan G_{best} dari semua partikel. G_{best} adalah nilai terbaik diantara semua partikel. P_{best} dan G_{best} adalah nilai bilangan

random yang dibangkitkan. Kemudian lakukan *update* untuk kecepatan awal dengan Rumus (2.21), dan Rumus (2.20). Kemudian dilakukan pengecekan apakah partikel setelah di *update* memenuhi batas bawah dan batas atas. Batas bawah adalah -1 dan batas atas adalah 1. Jika melewati batas bawah atau batas atas maka ubah nilai setiap partikel pada nilai batas bawah atau batas atas.

Dari posisi yang dihasilkan, lakukan pengurutan untuk masing-masing *container* 20 *feet* dan 40 *feet* untuk mendapatkan rute atau solusi pada masing-masing ukuran seperti sebelumnya sehingga akan menghasilkan solusi baru. Kemudian ulang langkah 5 sampai langkah 10 hingga kriteria penghentian terpenuhi.

11. Rubah posisi tumpukan

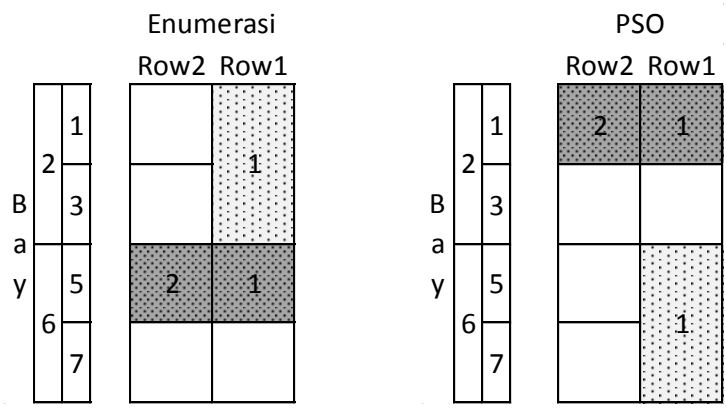
Dari hasil sebelumnya kalau masih mungkin untuk dilakaukan perubahan terhadap posisi *container* berdasarkan jumlah *tier* atau masih banyak kordinat posisi dalam kapal yang yang tidak terisi sampai tingkat atas, maka setiap tumpukan posisi yang belum penuh sampai jumlah *tier* berpeluang untuk ditempatkan *container* baru yang didapatkan dari kordinat posisi lain dengan tumpukan posisi paling sedikit. Kemudian diurutkan berdasarkan tujuan *container*, berat *container*. Sehingga menghilangkan kedua kendala tersebut dalam menghitung *penalty*. Kehudian dihitung fungsi tujuan dan dibandingkan dengan fungsi tujuan pada langkah 10, pilih nilai fungsi tujuan yang minimum.

3.4 Validasi

Dari pengembangan algoritma yang telah dilakukan sebelumnya dibuat program untuk menghitung fungsi tujuan. Dari hasil program dilakukan validasi untuk mengetahui apakah program yang telah dibuat sudah valid atau belum, validasi dilakukan dengan membandingkan hasil enumerasi dengan hasil program dari pengembangan algoritma pada kasus yang sangat sederhana, perbandingan hasil keduanya seperti pada Tabel 3.6 dan Gambar 3.12.

Tabel 3.6. Perbandingan Nilai Fungsi tujuan pada Enumerasi dan PSO

Metoda	**Enumerasi**	**PSO**
Penalti	0	0
Waktu *unloadig*	366	366
Fungsi tujuan	366	366



Gambar 3.12. Perbandingan Penataan Hasil Emunerasi dan PSO

Pada Tabel 3.6 dan Gambar 3.12 hasil algoritma *particle swarm optimization* dan enumerasi adalah sama. Oleh karena itu algoritma yang dikembangkan bisa dikatakan valid, sehingga penelitian dapat dilakukan ketahap selanjutnya.

BAB IV
EKSPERIMEN DAN ANALISIS

Dalam bab eksperimen dan analisis ini meliputi tahap pengujian algoritma dan implimentasi algoritma pada data yang telah dikumpulkan.

4.1 Pengujian Algoritma

Pengujian algoritma dibuat suatu program dengan bantuan software Matlab dan *output* dari program ini adalah fungsi tujuan yang didapat dari total *unloading* dan penalti, dan setiap total *unloading* didapatkan dari kordinat posisi setiap *container*, inti dari penelitan ini adalah mencari kordinat posisi setiap *container* dengan total *unloading* yang paling minimum dan penalti yang minimum sehingga nilai fungsi tujuan menjadi minimum.

Dalam bab ini dilakukan pengujian algoritma pada bermacam-macam data yang telah dikumpulkan. Berikut pengumpulan data, implimentasi algoritma dan analisis.

4.1.1 Pengumpulan data

Data yang digunakan merupakan data sekunder dari penelitian yang dilakukan Putamawa dan Santosa (2011), Putamawa dan Santosa membangkitkan data dengan bantuan software MATLAB dengan kriteria data dari penelitian Ambrosino et al (2004). Kriteria dari data seperti pada Tabel 4.1.

Tabel 4.1 Kriteria Data untuk Setiap Kasus

Kasus	Total kontainer	Ukuran		Berat (ton)			Tujuan		
		20'	40'	Light (10-15)	Medium (20-25)	Heavy (30-35)	d1	d2	d3
1	100	62	38	45	30	25	47	53	0
2	120	75	45	50	44	26	55	65	0
3	130	90	40	56	46	28	55	75	0
8	140	95	45	60	50	30	65	75	0
9	140	95	45	60	50	30	50	40	50

Dari kriteria Tabel 4.1 dibangkitkan data seperti pada lampiran B. dan jumlah *bay* =14, *row* =4, *tier*=5.

4.1.2 Implimentasi algoritma dan analisis.

Pengujian algoritma diimplimentasikan dengan lima macam kasus dengan parameter jumlah populasi atau N =50, θ_{min} =0.4, dan θ_{mak} =0.4, *stoping* kriteria adalah jumlah iterasi yaitu 100. Berikut hasil dari implimentasi algoritma dan perbandingannya dengan hasil penelitian sebelumnya dengan bermacam-macam jumlah data.

4.1.2.1 Kasus 1

Data yang digunakan untuk menguji algoritma pada kasus satu terdapat pada lampiran B.1. Berikut hasil dari percobaan PSO dibandingkan dengan BSO Modifikasi pada kasus 1 seperti pada Tabel 4.2.

Tabel 4.2 Perbandingan PSO dengan BSO Modifikasi pada Kasus 1.

Percobaan	PSO	BSO Modifikasi
1	14630	14488.8
2	14628	
3	14626	
4	14624	
5	14632	
6	14629	
7	14640	
8	14632	
9	14628	
10	14627	
Rata-rata	14629.6	
%Gap	0.97	
Terbaik	14624	
Gap	135.2	

Pada Tabel 4.2, pengujian algoritma PSO pada kasus 1 dilakukan percobaan sebanyak 10 kali percobaan, dan diperoleh rata-rata nilai fungsi tujuan 14.629,6 detik dan hasil terbaik adalah 14.624 detik. Sedangkan hasil dari BSO Modifikasi 14.488,8 detik. Kemudian dilakukan perbandingan antara kedua algoritma tersebut, dari perbandingan tersebut didapatkan %gap 0,97 persen, yang artinya hasil algoritma PSO lebih besar sebesar 0.97 persen jika dibandingkan dengan BSO Modifikasi, dan selisih antara nilai PSO terbaik dengan BSO Modifikasi adalah 135,2 detik. Pada kasus 1 BSO Modifikasi lebih baik dari PSO.

Kemudian dilakukan modifikasi pada PSO dengan menambahkan langkah pada algoritma PSO, yaitu hasil dari PSO awal kalau masih mungkin untuk dilakaukan perubahan terhadap posisi *container* berdasarkan jumlah *tier* atau masih banyak kordinat posisi dalam kapal yang yang tidak terisi sampai tingkat atas, maka setiap tumpukan posisi yang belum penuh sampai jumlah *tier* berpeluang untuk ditempatkan *container* baru yang didapatkan dari kordinat posisi lain dengan tumpukan posisi paling sedikit. Kemudian ditambahkan lagi langkah mengurut

tumpukan *container* berdasarkan tujuan *container* sehingga *container* dengan tujuan awal selalu ada diatas *container* dengan tujuan akhir. Dan ditambahkan lagi langkah mengurut tumpukan *container* berdasarkan jenis berat *container*. Dengan menambahkan ketiga langkah tersebut akan menjamin dua kendala yang pasti tidak akan dilanggar, yaitu kendala tujuan *container* dan kendala *container* yang lebih berat selalu ada dibawah *container* yang lebih ringan. Sehingga memungkinkan nilai penalti lebih kecih dan nilai fungsi tujuan menjadi minimum. Kemudian nilai fungsi tujuan baru dibandingkan dengan nilai fungsi tujuan sebelumnya, dan dipilih solusi dengan nilai fungsi tujuan terkecil. Berikut hasil dari percobaan PSO Modifikasi dibandingkan dengan BSO Modifikasi pada kasus 1 seperti pada Tabel 4.3.

Tabel 4.3 Perbandingan PSO Modifikasi dengan BSO Modifikasi pada Kasus 1.

Percobaan	PSO Modifikasi	BSO Modifikasi
1	14064	14488.8
2	14050	
3	14110	
4	14062	
5	14034	
6	14070	
7	14004	
8	14064	
9	14010	
10	14110	
Rata-rata	14057.8	
%Gap	-2.97	
Terbaik	14004	
Gap	-484.8	

Pada Tabel 4.3, pengujian algoritma PSO Modifikasi pada kasus 1 dilakukan percobaan sebanyak 10 kali percobaan, dan diperoleh rata-rata nilai fungsi tujuan 14.057,8 detik dan hasil terbaik adalah 14.004 detik. Sedangkan hasil dari BSO Modifikasi 14.488,8 detik. Kemudian dilakukan perbandingan antara kedua algoritma tersebut, dari perbandingan tersebut didapatkan %gap -2,97 persen, yang artinya hasil algoritma PSO Modifikasi lebih kecil sebesar 2.97 persen jika dibandingkan dengan BSO Modifikasi, dan selisih antara nilai PSO Modifikasi terbaik dengan BSO Modifikasi adalah 484,8 detik. Pada kasus 1 PSO Modifikasi lebih baik dari BSO Modifikasi. Kemudian dilakukan perbandingan pada PSO Modifikasi dangan solusi optimal dari metoda Heuristik. Berikut perbandingan PSO Modifikasi dengan solusi optimal dari metoda Heuristik, seperti pada Tabel 4.4.

Tabel 4.4 Perbandingan PSO Modifikasi dengan Solusi Optimal dari Metoda Heuristik pada Kasus 1.

Percobaan	PSO Modifikasi	Heuristik
1	14064	13854
2	14050	
3	14110	
4	14062	
5	14034	
6	14070	
7	14004	
8	14064	
9	14010	
10	14110	
Rata-rata	14057.8	
%Gap	1.47	
Terbaik	14004	
Gap	150	

Pada Tabel 4.4, pengujian algoritma PSO Modifikasi pada kasus 1 dilakukan percobaan sebanyak 10 kali percobaan, dan diperoleh rata-rata nilai fungsi tujuan 14.057,8 detik dan hasil terbaik adalah 14.004 detik. Sedangkan solusi optimal dari metoda Heuristik 13.854 detik. Kemudian dilakukan perbandingan antara kedua algoritma tersebut, dari perbandingan tersebut didapatkan %gap 1,47 persen, yang artinya hasil algoritma PSO Modifikasi lebih besar sebesar 2.97 persen jika dibandingkan dengan metoda Heuristik, dan selisih antara nilai PSO Modifikasi terbaik dengan metoda Heuristik adalah 150 detik. Pada kasus 1 metoda Heuristik lebih baik dari PSO Modifikasi. Kemudian dilakukan perbadingan PSO Modifikasi dengan BSO Modifikasi pada solusi optimal dari metoda Heuristik. Berikut perbandingan PSO Modifikasi dan BSO Modifikasi yang paling mendekati solusi optimal pada motede Heuristik, seperti pada Tabel 4.5.

Tabel 4.5 Perbandingan PSO Modifikasi dengan BSO Modifikasi yang Paling Mendekati Solusi Optimal dari Metoda Heuristik pada Kasus 1.

	BSO Modifikasi	Heuristik	PSO Modifikasi	Heuristik
%Gap	4.58		1.47	
Terbaik	14436		14004	
Gap	582		150	

Pada Tabel 4.5, perbandingan PSO Modifikasi dengan solusi optimal dari metoda Heuristik didapatkan %gap 1,47 persen, dan hasil terbaik adalah 14.004 detik, dan selisih antara nilai PSO Modifikasi terbaik dengan metoda Heuristik adalah 150 detik. Sedangkan pada perbandingan BSO Modifikasi dengan solusi optimal dari metoda Heuristik didapatkan %gap 4,58 persen, dan hasil terbaik adalah 14.436 detik, dan selisih antara nilai BSO Modifikasi terbaik dengan metoda Heuristik adalah 582 detik. Pada kasus 1 PSO Modifikasi lebih baik dari BSO Modifikasi.

4.1.2.2 Kasus 2

Data yang digunakan untuk menguji algoritma pada kasus satu terdapat pada lampiran B.2. Berikut hasil dari percobaan PSO dibandingkan dengan BSO Modifikasi pada kasus 2, seperti pada Tabel 4.6.

Tabel 4.6 Perbandingan PSO dengan BSO Modifikasi pada Kasus 2.

Percobaan	PSO	BSO Modifikasi
1	17287	17221.5
2	17306	
3	17298	
4	17297	
5	17294	
6	17306	
7	17289	
8	17279	
9	17304	
10	17282	
Rata-rata	17294.2	
%Gap	0.42	
Terbaik	17279	
Gap	57.5	

Pada Tabel 4.6, pengujian algoritma PSO pada kasus 2 dilakukan percobaan sebanyak 10 kali percobaan, dan diperoleh rata-rata nilai fungsi tujuan 17.294,2 detik dan hasil terbaik adalah 17.279 detik. Sedangkan hasil dari BSO Modifikasi 17.221,5 detik. Kemudian dilakukan perbandingan antara kedua algoritma tersebut, dari perbandingan tersebut didapatkan %gap 0,42 persen, yang artinya hasil algoritma PSO lebih besar sebesar 0.42 persen jika dibandingkan dengan BSO Modifikasi, dan selisih antara nilai PSO terbaik dengan BSO Modifikasi adalah 57,5 detik. Pada kasus 2 BSO Modifikasi lebih baik dari PSO.

Kemudian dilakukan modifikasi pada PSO dengan menambahkan langkah pada algoritma PSO, yaitu hasil dari PSO awal kalau masih mungkin untuk

dilakaukan perubahan terhadap posisi container berdasarkan jumlah tier atau masih banyak kordinat posisi dalam kapal yang yang tidak terisi sampai tingkat atas, maka setiap tumpukan posisi yang belum penuh sampai jumlah tier berpeluang untuk ditempatkan container baru yang didapatkan dari kordinat posisi lain dengan tumpukan posisi paling sedikit. Kemudian ditambahkan lagi langkah mengurut tumpukan container berdasarkan tujuan container sehingga container dengan tujuan awal selalu ada diatas container dengan tujuan akhir. Dan ditambahkan lagi langkah mengurut tumpukan container berdasarkan jenis berat container. Dengan menambahkan ketiga langkah tersebut akan menjamin dua kendala yang pasti tidak akan dilanggar, yaitu kendala tujuan container dan kendala container yang lebih berat selalu ada dibawah container yang lebih ringan. Sehingga memungkinkan nilai penalti lebih kecih dan nilai fungsi tujuan menjadi minimum. Kemudian nilai fungsi tujuan baru dibandingkan dengan nilai fungsi tujuan sebelumnya, dan dipilih solusi dengan nilai fungsi tujuan terkecil. Berikut hasil dari percobaan PSO Modifikasi dibandingkan dengan BSO Modifikasi pada kasus 2, seperti pada Tabel 4.7.

Tabel 4.7 Perbandingan PSO Modifikasi dengan BSO Modifikasi pada Kasus 2.

Percobaan	PSO Modifikasi	BSO Modifikasi
1	16930	17221.5
2	16950	
3	16800	
4	16830	
5	16860	
6	16960	
7	16740	
8	16830	
9	16800	
10	16770	
Rata-rata	16847	
%Gap	-2.17	
Terbaik	16740	
Gap	-481.5	

Pada Tabel 4.7, pengujian algoritma PSO Modifikasi pada kasus 2 dilakukan percobaan sebanyak 10 kali percobaan, dan diperoleh rata-rata nilai fungsi tujuan 16.847 detik dan hasil terbaik adalah 16.740 detik. Sedangkan hasil dari BSO Modifikasi 17.221,5 detik. Kemudian dilakukan perbandingan antara kedua algoritma tersebut, dari perbandingan tersebut didapatkan %gap -2,17 persen, yang artinya hasil algoritma PSO Modifikasi lebih kecil sebesar 2.17 persen jika dibandingkan dengan BSO Modifikasi, dan selisih antara nilai PSO Modifikasi terbaik dengan BSO Modifikasi adalah 481,5 detik. Pada kasus 2 PSO Modifikasi lebih baik dari BSO Modifikasi. Kemudian dilakukan perbandingan pada PSO Modifikasi dangan solusi optimal dari metoda Heuristik. Berikut perbandingan PSO Modifikasi dengan solusi optimal dari metoda Heuristik, seperti pada Tabel 4.8.

Tabel 4.8 Perbandingan PSO Modifikasi dengan Solusi Optimal dari Metoda Heuristik pada Kasus 2.

Percobaan	PSO Modifikasi	Heuristik
1	16930	16584
2	16950	
3	16800	
4	16830	
5	16860	
6	16960	
7	16740	
8	16830	
9	16800	
10	16770	
Rata-rata	16847	
%Gap	1.59	
Terbaik	16740	
Gap	156	

Pada Tabel 4.8, pengujian algoritma PSO Modifikasi pada kasus 2 dilakukan percobaan sebanyak 10 kali percobaan, dan diperoleh rata-rata nilai fungsi tujuan 16.847 detik dan hasil terbaik adalah 16.740 detik. Sedangkan solusi optimal dari metoda Heuristik 16.584 detik. Kemudian dilakukan perbandingan antara kedua algoritma tersebut, dari perbandingan tersebut didapatkan %gap 1,59 persen, yang artinya hasil algoritma PSO Modifikasi lebih besar sebesar 1,59 persen jika dibandingkan dengan metoda Heuristik, dan selisih antara nilai PSO Modifikasi terbaik dengan metoda Heuristik adalah 156 detik. Pada kasus 2 metoda Heuristik lebih baik dari PSO Modifikasi. Kemudian dilakukan perbadingan PSO Modifikasi dengan BSO Modifikasi pada solusi optimal dari metoda Heuristik. Berikut perbandingan PSO Modifikasi dan BSO Modifikasi yang paling mendekati solusi optimal pada moteda Heuristik, seperti pada Tabel 4.9.

Tabel 4.9 Perbandingan PSO Modifikasi dengan BSO Modifikasi yang Paling Mendekati Solusi Optimal dari Metoda Heuristik pada Kasus 2.

	BSO Modifikasi	Heuristik	PSO Modifikasi	Heuristik
%Gap	3.84		1.59	
Terbaik	17190		16740	
Gap	606		156	

Pada Tabel 4.9, perbandingan PSO Modifikasi dengan solusi optimal dari metoda Heuristik didapatkan %gap 1,59 persen, dan hasil terbaik adalah 16.740 detik, dan selisih antara nilai PSO Modifikasi terbaik dengan metoda Heuristik adalah 156 detik. Sedangkan pada perbandingan BSO Modifikasi dengan solusi optimal dari metoda Heuristik didapatkan %gap 3,84 persen, dan hasil terbaik adalah 16.740 detik, dan selisih antara nilai BSO Modifikasi terbaik dengan metoda Heuristik adalah 606 detik. Pada kasus 2 PSO Modifikasi lebih baik dari BSO Modifikasi.

4.1.2.3 Kasus 3

Data yang digunakan untuk menguji algoritma pada kasus satu terdapat pada lampiran B.3. Berikut hasil dari percobaan PSO dibandingkan dengan BSO Modifikasi pada kasus 3, seperti pada Tabel 4.10.

Tabel 4.10 Perbandingan PSO dengan BSO Modifikasi pada Kasus 3.

Percobaan	PSO	BSO Modifikasi
1	18909	18504.0
2	18878	
3	18869	
4	18883	
5	18867	
6	18866	
7	18879	
8	18883	
9	18880	
10	18879	
Rata-rata	18879.3	
%Gap	2.03	
Terbaik	18866	
Gap	362.0	

Pada Tabel 4.10, pengujian algoritma PSO pada kasus 3 dilakukan percobaan sebanyak 10 kali percobaan, dan diperoleh rata-rata nilai fungsi tujuan 18.879,3 detik dan hasil terbaik adalah 18.866 detik. Sedangkan hasil dari BSO Modifikasi 18.504 detik. Kemudian dilakukan perbandingan antara kedua algoritma tersebut, dari perbandingan tersebut didapatkan %gap 2,03 persen, yang artinya hasil algoritma PSO lebih besar sebesar 2,03 persen jika dibandingkan dengan BSO Modifikasi, dan selisih antara nilai PSO terbaik dengan BSO Modifikasi adalah 362 detik. Pada kasus 3 BSO Modifikasi lebih baik dari PSO.

Kemudian dilakukan modifikasi pada PSO dengan menambahkan langkah pada algoritma PSO, yaitu hasil dari PSO awal kalau masih mungkin untuk dilakaukan perubahan terhadap posisi container berdasarkan jumlah tier atau masih banyak kordinat posisi dalam kapal yang yang tidak terisi sampai tingkat atas, maka setiap tumpukan posisi yang belum penuh sampai jumlah tier berpeluang untuk ditempatkan container baru yang didapatkan dari kordinat posisi lain dengan tumpukan posisi paling sedikit. Kemudian ditambahkan lagi langkah mengurut tumpukan container berdasarkan tujuan container sehingga container dengan tujuan

awal selalu ada diatas container dengan tujuan akhir. Dan ditambahkan lagi langkah mengurut tumpukan container berdasarkan jenis berat container. Dengan menambahkan ketiga langkah tersebut akan menjamin dua kendala yang pasti tidak akan dilanggar, yaitu kendala tujuan container dan kendala container yang lebih berat selalu ada dibawah container yang lebih ringan. Sehingga memungkinkan nilai penalti lebih kecih dan nilai fungsi tujuan menjadi minimum. Kemudian nilai fungsi tujuan baru dibandingkan dengan nilai fungsi tujuan sebelumnya, dan dipilih solusi dengan nilai fungsi tujuan terkecil. Berikut hasil dari percobaan PSO Modifikasi dibandingkan dengan BSO Modifikasi pada kasus 3, seperti pada Tabel 4.11.

Tabel 4.11 Perbandingan PSO Modifikasi dengan BSO Modifikasi pada Kasus 3.

Percobaan	PSO Modifikasi	BSO Modifikasi
1	18190	18504.0
2	17940	
3	17910	
4	18060	
5	17910	
6	17970	
7	17940	
8	17970	
9	18190	
10	18130	
Rata-rata	18021	
%Gap	-2.61	
Terbaik	17910	
Gap	-594.0	

Pada Tabel 4.11, pengujian algoritma PSO Modifikasi pada kasus 3 dilakukan percobaan sebanyak 10 kali percobaan, dan diperoleh rata-rata nilai fungsi tujuan 18.021 detik dan hasil terbaik adalah 17.910 detik. Sedangkan hasil dari BSO Modifikasi 18.504 detik. Kemudian dilakukan perbandingan antara kedua algoritma tersebut, dari perbandingan tersebut didapatkan %gap -2,61 persen, yang

artinya hasil algoritma PSO Modifikasi lebih kecil sebesar 2,61 persen jika dibandingkan dengan BSO Modifikasi, dan selisih antara nilai PSO Modifikasi terbaik dengan BSO Modifikasi adalah 594 detik. Pada kasus 3 PSO Modifikasi lebih baik dari BSO Modifikasi. Kemudian dilakukan perbandingan pada PSO Modifikasi dangan solusi optimal dari metoda Heuristik. Berikut perbandingan PSO Modifikasi dengan solusi optimal dari metoda Heuristik, seperti pada Tabel 4.12.

Tabel 4.12 Perbandingan PSO Modifikasi Dengan Solusi Optimal dari Metoda Heuristik pada Kasus 3.

Percobaan	PSO Modifikasi	Heuristik
1	18190	18096
2	17940	
3	17910	
4	18060	
5	17910	
6	17970	
7	17940	
8	17970	
9	18190	
10	18130	
Rata-rata	18021	
%Gap	-0.41	
Terbaik	17910	
Gap	-186	

Pada Tabel 4.12, pengujian algoritma PSO Modifikasi pada kasus 3 dilakukan percobaan sebanyak 10 kali percobaan, dan diperoleh rata-rata nilai fungsi tujuan 18.021 detik dan hasil terbaik adalah 17.910 detik. Sedangkan solusi optimal dari metoda Heuristik 18.096 detik. Kemudian dilakukan perbandingan antara kedua algoritma tersebut, dari perbandingan tersebut didapatkan %gap -0,41 persen, yang artinya hasil algoritma PSO Modifikasi lebih kecil sebesar 0,41 persen jika dibandingkan dengan metoda Heuristik, dan selisih antara nilai PSO Modifikasi

terbaik dengan metoda Heuristik adalah 186 detik. Pada kasus 3 PSO Modifikasi lebih baik dari metoda Heuristik. Kemudian dilakukan perbadingan PSO Modifikasi dengan BSO Modifikasi pada solusi optimal dari metoda Heuristik. Berikut perbandingan PSO Modifikasi dan BSO Modifikasi yang paling mendekati solusi optimal pada moteda Heuristik, seperti pada Tabel 4.13.

Tabel 4.13 Perbandingan PSO Modifikasi dengan BSO Modifikasi yang Paling Mendekati Solusi Optimal dari Metoda Heuristik pada Kasus 3.

	BSO Modifikasi	Heuristik	PSO Modifikasi	Heuristik
%Gap	2.25		-0.41	
Terbaik	18456		17910	
Gap	360		-186	

Pada Tabel 4.13, perbandingan PSO Modifikasi dengan solusi optimal dari metoda Heuristik didapatkan %gap -0,41 persen, dan hasil terbaik adalah 17.910 detik, dan selisih antara nilai PSO Modifikasi terbaik dengan metoda Heuristik adalah 186 detik. Sedangkan pada perbandingan BSO Modifikasi dengan solusi optimal dari metoda Heuristik didapatkan %gap 2,25 persen, dan hasil terbaik adalah 18.456 detik, dan selisih antara nilai BSO Modifikasi terbaik dengan metoda Heuristik adalah 360 detik. Pada kasus 3 PSO Modifikasi lebih baik dari BSO Modifikasi.

4.1.2.4 Kasus 4

Data yang digunakan untuk menguji algoritma pada kasus satu terdapat pada lampiran B.4. Berikut hasil dari percobaan PSO dibandingkan dengan BSO Modifikasi pada kasus 4, seperti pada Tabel 4.14.

Tabel 4.14 Perbandingan PSO dengan BSO Modifikasi pada Kasus 4.

Percobaan	PSO	BSO Modifikasi
1	20300	19857.0
2	20321	
3	20305	
4	20315	
5	20300	
6	20301	
7	20299	
8	20297	
9	20300	
10	20305	
Rata-rata	20304.3	
%Gap	2.25	
Terbaik	20297	
Gap	440.0	

Pada Tabel 4.14, pengujian algoritma PSO pada kasus 4 dilakukan percobaan sebanyak 10 kali percobaan, dan diperoleh rata-rata nilai fungsi tujuan 20.304,3 detik dan hasil terbaik adalah 20.297 detik. Sedangkan hasil dari BSO Modifikasi 19.857 detik. Kemudian dilakukan perbandingan antara kedua algoritma tersebut, dari perbandingan tersebut didapatkan %gap 2,25 persen, yang artinya hasil algoritma PSO lebih besar sebesar 2,25 persen jika dibandingkan dengan BSO Modifikasi, dan selisih antara nilai PSO terbaik dengan BSO Modifikasi adalah 440 detik. Pada kasus 4 BSO Modifikasi lebih baik dari PSO.

Kemudian dilakukan modifikasi pada PSO dengan menambahkan langkah pada algoritma PSO, yaitu hasil dari PSO awal kalau masih mungkin untuk dilakaukan perubahan terhadap posisi *container* berdasarkan jumlah *tier* atau masih banyak kordinat posisi dalam kapal yang yang tidak terisi sampai tingkat atas, maka setiap tumpukan posisi yang belum penuh sampai jumlah *tier* berpeluang untuk ditempatkan *container* baru yang didapatkan dari kordinat posisi lain dengan tumpukan posisi paling sedikit. Kemudian ditambahkan lagi langkah mengurut tumpukan *container* berdasarkan tujuan *container* sehingga *container* dengan

tujuan awal selalu ada diatas *container* dengan tujuan akhir. Dan ditambahkan lagi langkah mengurut tumpukan *container* berdasarkan jenis berat *container*. Dengan menambahkan ketiga langkah tersebut akan menjamin dua kendala yang pasti tidak akan dilanggar, yaitu kendala tujuan *container* dan kendala *container* yang lebih berat selalu ada dibawah *container* yang lebih ringan. Sehingga memungkinkan nilai penalti lebih kecih dan nilai fungsi tujuan menjadi minimum. Kemudian nilai fungsi tujuan baru dibandingkan dengan nilai fungsi tujuan sebelumnya, dan dipilih solusi dengan nilai fungsi tujuan terkecil. Berikut hasil dari percobaan PSO Modifikasi dibandingkan dengan BSO Modifikasi pada kasus 4, seperti pada Tabel 4.15.

Tabel 4.15 Perbandingan PSO Modifikasi dengan BSO Modifikasi pada Kasus 4.

Percobaan	PSO Modifikasi	BSO Modifikasi
1	19620	19857.0
2	19710	
3	19530	
4	19650	
5	19620	
6	19690	
7	19690	
8	19620	
9	19620	
10	19530	
Rata-rata	19628	
%Gap	-1.15	
Terbaik	19530	
Gap	-327.0	

Pada Tabel 4.15, pengujian algoritma PSO Modifikasi pada kasus 4 dilakukan percobaan sebanyak 10 kali percobaan, dan diperoleh rata-rata nilai fungsi tujuan 19.628 detik dan hasil terbaik adalah 19.530 detik. Sedangkan hasil

dari BSO Modifikasi 19.857 detik. Kemudian dilakukan perbandingan antara kedua algoritma tersebut, dari perbandingan tersebut didapatkan %gap -1,15 persen, yang artinya hasil algoritma PSO Modifikasi lebih kecil sebesar 1,15 persen jika dibandingkan dengan BSO Modifikasi, dan selisih antara nilai PSO Modifikasi terbaik dengan BSO Modifikasi adalah 327 detik. Pada kasus 4 PSO Modifikasi lebih baik dari BSO Modifikasi. Kemudian dilakukan perbandingan pada PSO Modifikasi dangan solusi optimal dari metoda Heuristik. Berikut perbandingan PSO Modifikasi dengan solusi optimal dari metoda Heuristik, seperti pada Tabel 4.16.

Tabel 4.16 Perbandingan PSO Modifikasi dengan Solusi Optimal dari Metoda Heuristik pada Kasus 4.

Percobaan	PSO Modifikasi	Heuristik
1	19620	19440
2	19710	
3	19530	
4	19650	
5	19620	
6	19690	
7	19690	
8	19620	
9	19620	
10	19530	
Rata-rata	19628	
%Gap	0.97	
Terbaik	19530	
Gap	90	

Pada Tabel 4.16, pengujian algoritma PSO Modifikasi pada kasus 4 dilakukan percobaan sebanyak 10 kali percobaan, dan diperoleh rata-rata nilai fungsi tujuan 19.628 detik dan hasil terbaik adalah 19.530 detik. Sedangkan solusi optimal dari metoda Heuristik 13.854 detik. Kemudian dilakukan perbandingan

antara kedua algoritma tersebut, dari perbandingan tersebut didapatkan %gap 0,97 persen, yang artinya hasil algoritma PSO Modifikasi lebih besar sebesar 0.97 persen jika dibandingkan dengan metoda Heuristik, dan selisih antara nilai PSO Modifikasi terbaik dengan metoda Heuristik adalah 90 detik. Pada kasus 4 metoda Heuristik lebih baik dari PSO Modifikasi. Kemudian dilakukan perbadingan PSO Modifikasi dengan BSO Modifikasi pada solusi optimal dari metoda Heuristik. Berikut perbandingan PSO Modifikasi dan BSO Modifikasi yang paling mendekati solusi optimal pada moteda Heuristik, seperti pada Tabel 4.17.

Tabel 4.17 Perbandingan PSO Modifikasi dengan BSO Modifikasi yang Paling Mendekati Solusi Optimal dari Metoda Heuristik pada Kasus 4.

	BSO Modifikasi	Heuristik	PSO Modifikasi	Heuristik
%Gap	2.15		0.97	
Terbaik	19818		19530	
Gap	378		90	

Pada Tabel 4.17, perbandingan PSO Modifikasi dengan solusi optimal dari metoda Heuristik didapatkan %gap 0,97 persen, dan hasil terbaik adalah 19.530 detik, dan selisih antara nilai PSO Modifikasi terbaik dengan metoda Heuristik adalah 90 detik. Sedangkan pada perbandingan BSO Modifikasi dengan solusi optimal dari metoda Heuristik didapatkan %gap 2,15 persen, dan hasil terbaik adalah 19.818 detik, dan selisih antara nilai BSO Modifikasi terbaik dengan metoda Heuristik adalah 378 detik. Pada kasus 4 PSO Modifikasi lebih baik dari BSO Modifikasi.

4.1.2.5 Kasus 5

Data yang digunakan untuk menguji algoritma pada kasus satu terdapat pada lampiran B.5. Berikut hasil dari percobaan PSO dibandingkan dengan BSO Modifikasi pada kasus 5, seperti pada Tabel 4.18.

Tabel 4.18 Perbandingan PSO dengan BSO Modifikasi pada Kasus 5.

Percobaan	PSO	BSO Modifikasi
1	20310	19842.0
2	20313	
3	20310	
4	20306	
5	20307	
6	20304	
7	20307	
8	20319	
9	20307	
10	20306	
Rata-rata	20308.9	
%Gap	2.35	
Terbaik	20304	
Gap	462.0	

Pada Tabel 4.18, pengujian algoritma PSO pada kasus 5 dilakukan percobaan sebanyak 10 kali percobaan, dan diperoleh rata-rata nilai fungsi tujuan 20.308,9 detik dan hasil terbaik adalah 20.304 detik. Sedangkan hasil dari BSO Modifikasi 19.842 detik. Kemudian dilakukan perbandingan antara kedua algoritma tersebut, dari perbandingan tersebut didapatkan %gap 2,35 persen, yang artinya hasil algoritma PSO lebih besar sebesar 2,35 persen jika dibandingkan dengan BSO Modifikasi, dan selisih antara nilai PSO terbaik dengan BSO Modifikasi adalah 462 detik. Pada kasus 5 BSO Modifikasi lebih baik dari PSO.

Kemudian dilakukan modifikasi pada PSO dengan menambahkan langkah pada algoritma PSO, yaitu hasil dari PSO awal kalau masih mungkin untuk

dilakaukan perubahan terhadap posisi *container* berdasarkan jumlah *tier* atau masih banyak kordinat posisi dalam kapal yang yang tidak terisi sampai tingkat atas, maka setiap tumpukan posisi yang belum penuh sampai jumlah *tier* berpeluang untuk ditempatkan *container* baru yang didapatkan dari kordinat posisi lain dengan tumpukan posisi paling sedikit. Kemudian ditambahkan lagi langkah mengurut tumpukan *container* berdasarkan tujuan *container* sehingga *container* dengan tujuan awal selalu ada diatas *container* dengan tujuan akhir. Dan ditambahkan lagi langkah mengurut tumpukan *container* berdasarkan jenis berat *container*. Dengan menambahkan ketiga langkah tersebut akan menjamin dua kendala yang pasti tidak akan dilanggar, yaitu kendala tujuan *container* dan kendala *container* yang lebih berat selalu ada dibawah *container* yang lebih ringan. Sehingga memungkinkan nilai penalti lebih kecih dan nilai fungsi tujuan menjadi minimum. Kemudian nilai fungsi tujuan baru dibandingkan dengan nilai fungsi tujuan sebelumnya, dan dipilih solusi dengan nilai fungsi tujuan terkecil. Berikut hasil dari percobaan PSO Modifikasi dibandingkan dengan BSO Modifikasi pada kasus 5, seperti pada Tabel 4.19.

Tabel 4.19 Perbandingan PSO Modifikasi dengan BSO Modifikasi pada Kasus 5.

Percobaan	PSO Modifikasi	BSO Modifikasi
1	19560	19842.0
2	19560	
3	19560	
4	19650	
5	19650	
6	19690	
7	19650	
8	19620	
9	19630	
10	19620	
Rata-rata	19619	
%Gap	-1.12	
Terbaik	19560	
Gap	-282.0	

Pada Tabel 4.19, pengujian algoritma PSO Modifikasi pada kasus 5 dilakukan percobaan sebanyak 10 kali percobaan, dan diperoleh rata-rata nilai fungsi tujuan 19.619 detik dan hasil terbaik adalah 19.560 detik. Sedangkan hasil dari BSO Modifikasi 19.842 detik. Kemudian dilakukan perbandingan antara kedua algoritma tersebut, dari perbandingan tersebut didapatkan %gap -1,12 persen, yang artinya hasil algoritma PSO Modifikasi lebih kecil sebesar 1,12 persen jika dibandingkan dengan BSO Modifikasi, dan selisih antara nilai PSO Modifikasi terbaik dengan BSO Modifikasi adalah 282 detik. Pada kasus 5 PSO Modifikasi lebih baik dari BSO Modifikasi. Kemudian dilakukan perbandingan pada PSO Modifikasi dangan solusi optimal dari metoda Heuristik. Berikut perbandingan PSO Modifikasi dengan solusi optimal dari metoda Heuristik, seperti pada Tabel 4.20.

Tabel 4.20 Perbandingan PSO Modifikasi dengan Solusi Optimal dari Metoda Heuristik pada Kasus 5.

Percobaan	PSO Modifikasi	Heuristik
1	19560	19470
2	19560	
3	19560	
4	19650	
5	19650	
6	19690	
7	19650	
8	19620	
9	19630	
10	19620	
Rata-rata	19619	
%Gap	0.77	
Terbaik	19560	
Gap	90	

Pada Tabel 4.20, pengujian algoritma PSO Modifikasi pada kasus 5 dilakukan percobaan sebanyak 10 kali percobaan, dan diperoleh rata-rata nilai fungsi tujuan 19.619 detik dan hasil terbaik adalah 19.560 detik. Sedangkan solusi optimal dari metoda Heuristik 19.470 detik. Kemudian dilakukan perbandingan antara kedua algoritma tersebut, dari perbandingan tersebut didapatkan %gap 0,77 persen, yang artinya hasil algoritma PSO Modifikasi lebih besar sebesar 0,77 persen jika dibandingkan dengan metoda Heuristik, dan selisih antara nilai PSO Modifikasi terbaik dengan metoda Heuristik adalah 90 detik. Pada kasus 5 metoda Heuristik lebih baik dari PSO Modifikasi. Kemudian dilakukan perbadingan PSO Modifikasi dengan BSO Modifikasi pada solusi optimal dari metoda Heuristik. Berikut perbandingan PSO Modifikasi dan BSO Modifikasi yang paling mendekati solusi optimal pada moteda Heuristik, seperti pada Tabel 4.21.

Tabel 4.21 Perbandingan PSO Modifikasi dengan BSO Modifikasi yang Paling Mendekati Solusi Optimal dari Metoda Heuristik pada Kasus 5.

	BSO Modifikasi	Heuristik	PSO Modifikasi	Heuristik
%Gap	2.08		0.77	
Terbaik	19842		19560	
Gap	372		90	

Pada Tabel 4.21, perbandingan PSO Modifikasi dengan solusi optimal dari metoda Heuristik didapatkan %gap 0,77 persen, dan hasil terbaik adalah 19.560 detik, dan selisih antara nilai PSO Modifikasi terbaik dengan metoda Heuristik adalah 90 detik. Sedangkan pada perbandingan BSO Modifikasi dengan solusi optimal dari metoda Heuristik didapatkan %gap 2,08 persen, dan hasil terbaik adalah 19.842 detik, dan selisih antara nilai BSO Modifikasi terbaik dengan metoda Heuristik adalah 372 detik. Pada kasus 5 PSO Modifikasi lebih baik dari BSO Modifikasi.

4.2 Analisis validasi

Pada proses validasi algoritma dilakukan dengan menyelesaikan kasus sederhana, kasus sederhana ini diselesaikan dengan dua metoda, yaitu teknik enumerasi dan algoritma *Particle Swarm Optimization* (PSO) yang telah dimodifikasi dan disesuaikan pada kasus *Container Stowage Problem* (CSP). Proses validasi dilakukan untuk menjamin bahwa hasil dari modifikasi PSO sudah benar dalam menyelesaikan permasalahan CSP dan mampu menemukan solusi optimal. Algoritma dikatakan valid ketika algoritma yang diusulkan mampu menghasilkan solusi yang sama dengan solusi optimal yang didapat dari teknik enumerasi.

Dari proses validasi yang telah dilakukan, hasil dari algoritma PSO yang telah disesuaikan untuk menyelesaikan permasalahan CSP sama dengan solusi optimal dengan teknik enumerasi, maka dari itu algoritma yang telah dikembangkan dikatakan valid karena sudah mampu menghasilkan solusi optimal pada kasus sederhana. Tetapi untuk menguji kemampuan algoritma untuk semua kasus perlu dilakukannya eksperimen terhadap kasus yang lebih besar.

4.3 Analisis *performance* algoritma

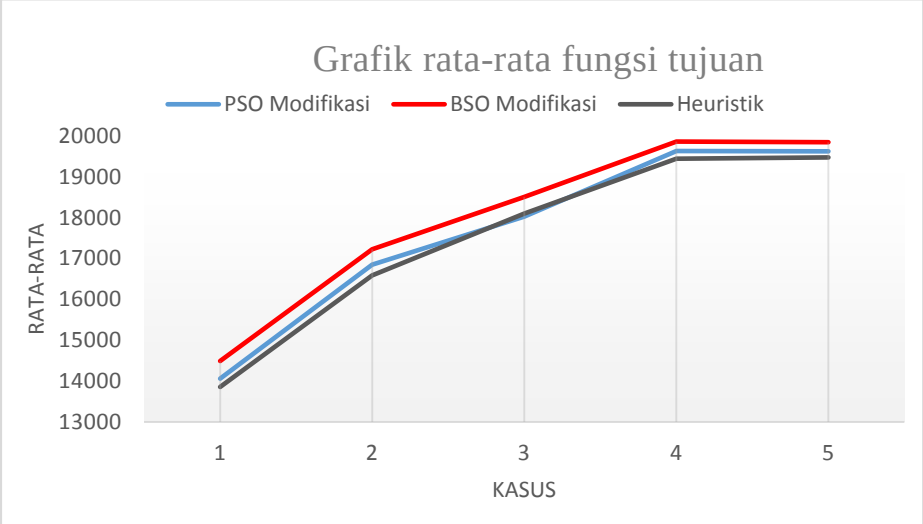
Algoritma *Particle Swarm Optimization* (PSO) untuk *Container Stowage Problem* (CSP) telah dilakukan modifikasi atau penambahan langkah-langkah dalam menyelesaikan permasalahan CSP, permasalahan CSP meupakan permasalahan penataan peti kemas kedalam kapal yang termasuk *NP-Hard Problem* dan susah dipecahkan secara eksak karena banyak memiliki fungsi kendala dan bahkan antara fungsi kendala terjadi konflikting, sehingga memungkinkan sangat susah untuk dipecahkan secara manual. Untuk mengatasi fungsi kendala ini algoritma PSO ditambahkan penalti, penalti digunakan untuk mengontrol sampel dalam menemukan solusi optimal. Setiap sampel yang terkena penalti akan ditambahkan dengan nilai fungsi tujuan pada sampel tersebut sehingga nilai fungsi tujuan pada sampel tersebut akan menjadi besar dan memungkinkan untuk iterasi selanjutnya sampel tersebut tidak akan terpilih lagi sebagai kawanan populasi dalam PSO. Pada modifikasi PSO untuk kasus CSP pembangkitan

kordinat posisi *container* atau peti kemas dibangktikan dengan bilangan random, dan ditambahkan langkah mengurut *container* berdasarkan berat *container*, kemudian hasil dari langkah pertama kalau masih mungkin untuk dilakaukan perubahan terhadap posisi *container* berdasarkan banyaknya kordinat posisi dalam kapal yang yang tidak terisi sampai tingkat atas, sehingga setiap posisi yang belum penuh sampai jumlah tingkat atas berpeluang untuk ditempatkan *container* baru yang didapatkan dari kordinat posisi lain dengan tumpukan posisi paling sedikit. Kemudian diurut berdasarkan tujuan *container* sehingga *container* dengan tujuan awal selalu ada diatas *container* dengan tujuan akhir. Dan ditambahkan lagi langkah mengurut tumpukan *container* berdasarkan jenis berat *container*. Dari modifikasi PSO untuk CSP dilakukan eksperimen terhadap lima macam kasus. Kelima macam kasus tersebut diaplikasikan pada PSO, dan PSO Modifikasi dibandingkan dengan BSO Modifikasi pada penelitian sebelumnya.

Dari kelima macam kasus tersebut didapatkan PSO masih belum mampu menemukan solusi optimal, hal ini dapat dilihat dari hasil %Gap dan Gap menunjukkan nilai PSO masih lebih besar dari BSO Modifikasi, sehingga dilakukan modifikasi pada PSO.

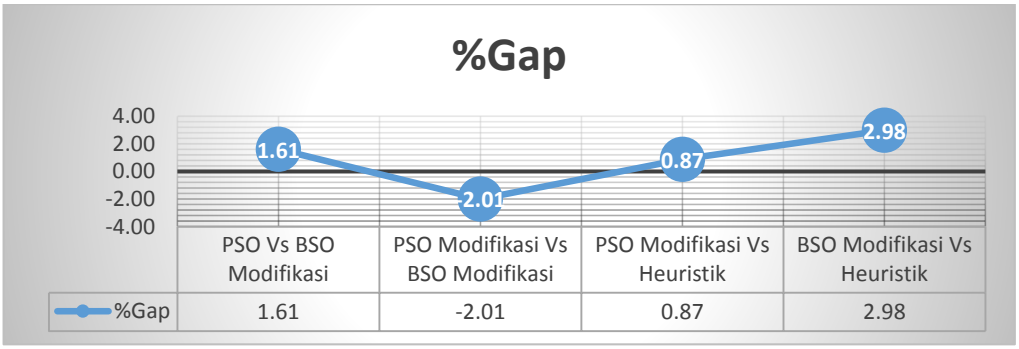
Tabel 4.22 Perbandingan Algoritma

Perbandingan	Kasus	Rata-rata	%Gap	Terbaik	Gap
PSO Vs BSO Modifikasi	1	14629.60	0.97	14624	135.2
	2	17294.20	0.42	17279	57.5
	3	18879.30	2.03	18866	362.0
	4	20304.30	2.25	20297	440.0
	5	20308.90	2.35	20304	462.0
PSO Modifikasi Vs BSO Modifikasi	1	14057.80	-2.97	14004	-484.8
	2	16847.00	-2.17	16740	-481.5
	3	18021.00	-2.61	17910	-594.0
	4	19628.00	-1.15	19530	-327.0
	5	19619.00	-1.12	19560	-282.0
PSO Modifikasi Vs Heuristik	1	14057.80	1.47	14004	150.0
	2	16847.00	1.59	16740	156.0
	3	18021.00	-0.41	17910	-186.0
	4	19628.00	0.97	19530	90.0
	5	19619.00	0.77	19560	90.0
BSO Modifikasi Vs Heuristik	1	14488.80	4.58	14436	582.0
	2	17221.50	3.84	17190	606.0
	3	18504.00	2.25	18456	360.0
	4	19857.00	2.15	19818	378.0
	5	19875.00	2.08	19842	372.0

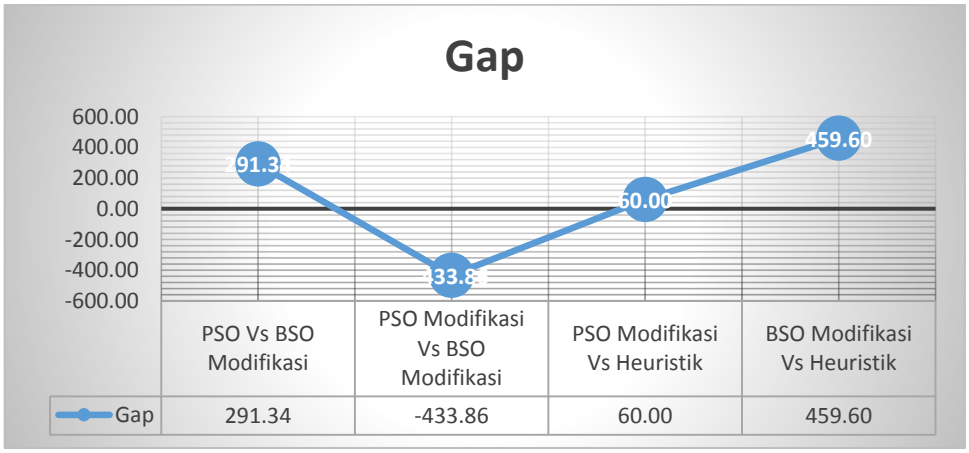


Gambar 4.1 Grafik Rata-Rata Fungsi Tujuan

Pada Gambar 4.1 rata-rata fungsi tujuan yang paling besar dengan menggunakan algoritma BSO Modifikasi, sedangkan algoritma PSO Modifikasi pada kasus 1, 2, 4, dan 5 rata-ratanya hampir sama dengan solusi optimal pada metoda heuristik, pada kasus 3 algoritma PSO Modifikasi lebih kecil dari metoda heuristik.



Gambar 4.2 %Gap untuk Semua Perbadingan Algoritma



Gambar 4.3 Gap untuk Semua Perbadingan Algoritma

Pada Tabel 4.22 perbandingan PSO dengan BSO Modifikasi semua nilai %Gap bernilai positif yang berarti nilai dari PSO selalu lebih besar dibandingkan

dengan BSO Modifikasi untuk semua kasus. Namun dengan dilakukannya modifikasi pada PSO semua nilai %Gap bernilai negative yang berarti nilai dari PSO Modifikasi selalu lebih kecil atau lebih baik dari nilai BSO Modifikasi. Seperti pada Gambar 4.2 dan Gambar 4.3 nilai rata-rata %Gap dan Gap untuk semua kasus PSO Modifikasi lebih kecil sebesar 2,01 persen atau 433,86 detik dibandingkan dengan BSO Modifikasi. Pada perbandingan PSO Modifikasi dengan solusi optimal dari metoda Heuristik untuk semua kasus seperti pada Gambar 4.2 dan 4.3 nilai rata-rata %Gap 0.87 atau 60 detik, angka ini hampir mendekani nilai optimal pada metoda Heuristik, bahkan pada kasus 3 PSO Modifikasi lebih rendah dibandingkan Heuristik sebesar 0.41 persen atau 186 detik. Nilai ini lebih rendah dari perbandingan BSO Modifikasi dengan Heuristik yaitu nilai rata-rata %Gap 2,98 atau 459,6 detik. Sehingga PSO Modifikasi lebih baik dari BSO Modifikasi.

BAB V
KESIMPULAN DAN SARAN

Pada bab ini berisi kesimpulan dari seluruh tahapan penelitian dan saran yang berkaitan dengan penelitian selanjutnya yang bersesuaian dengan topik permasalahan pada penelitian ini.

5.1 Kesimpulan

Proses validasi pada modifikasi algoritma *Particle Swarm Optimization* (PSO) dilakuakan dengan membandingkan hasil dari modifikasi PSO dengan teknik enumerasi yang pasti mendapatkan solusi optimal pada kasus yang sederhana. Hasil dari algoritma yang dikembangkan dapat disimpulkan valid karena dapat menghasilkan nilai yang sama dengan teknik enumerasi.

Fungsi kendala diwakili dengan menambahkan penalti. Modifikasi pada PSO dengan menambahkan langkah pada algoritma PSO, yaitu perubahan terhadap posisi *container* berdasarkan jumlah *tier*, mengurut tumpukan *container* berdasarkan tujuan *container,* dan mengurut tumpukan *container* berdasarkan jenis berat *container*. Dengan menambahkan ketiga langkah tersebut akan menjamin dua kendala yang pasti tidak akan dilanggar, yaitu kendala tujuan *container* dan kendala *container* yang lebih berat selalu ada dibawah *container* yang lebih ringan. Sehingga memungkinkan nilai penalti lebih kecil dan nilai fungsi tujuan menjadi minimum. Pengembangan algoritma telah dilakukan eksperimen pada lima macam kasus. Pebandingan PSO modifikasi dengan BSO modifikasi didapatkan %Gap -2.01 persen yang berarti PSO modifikasi memberikan nilai yang lebih kecil sebesar 2.01 persen atau 433.86 detik jika dibandingkan dengan BSO modifikasi. Kemudian pebandingan PSO modifikasi dengan Heuristik didapatkan %Gap 0.87 yang berarti PSO modifikasi memberikan nilai yang lebih besar sebesar 0.87 persen atau 60 detik jika dibandingkan dengan Heuristik.

5.2 Saran

Untuk penelitian selanjutnya, minimasi waktu tidak hanya untuk proses *unloading* saja, tetapi dengan mempertimbangkan proses waktu *loading*, serta kasus yang digunakan tidak hanya dua jenis ukuran *container*.

DAFTAR PUSTAKA

Ambrosino, D., Anghinolfi, D., Paolucci, M., & Sciomachen, A. (2010). An experimental comparison of different heuristics for the master *bay* plan problem. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 6049 LNCS*, 314–325.

Ambrosino, D., Paolucci, M., & Sciomachen, A. (2015). A MIP Heuristic for Multi Port Stowage Planning. *Transportation Research Procedia*, *10*(July), 725–734.

Ambrosino, D., Sciomachen, A., & Tanfani, E. (2004). Stowing a *container*ship: The master *bay* plan problem. *Transportation Research Part A: Policy and Practice*, *38*(2), 81–99.

Araujo, E. J., Chaves, A. A., De Salles Neto, L. L., & De Azevedo, A. T. (2016). Pareto clustering search applied for 3D *container* ship loading plan problem. *Expert Systems with Applications*, *44*, 50–57.

Avriel, M., Penn, M., & Shpirer, N. (2000). *Container* ship stowage problem: complexity and connection to the coloring of circle graphs. *Discrete Applied Mathematics*, *103*(1-3), 271–279.

Delgado, A., Jensen, R. M., Janstrup, K., Rose, T. H., & Andersen, K. H. (2012). A Constraint Programming model for fast optimal stowage of *container* vessel *bay*s. *European Journal of Operational Research*, *220*(1), 251–261.

Ding, D., & Chou, M. C. (2015). Stowage planning for *container* ships: A heuristic algorithm to reduce the number of shifts. *European Journal of Operational Research*, *246*(1), 242–249.

Dubrovsky, O., Levitin, G., & Penn, M. (2002). A genetic algorithm with a compact solution encoding for the *container* ship stowage problem. *Journal of Heuristics*, *8*(6), 585–599.

Fan, L., Low, M., Ying, H., & Jing, H. (2010). Stowage Planning of Large *Container*ship with Tradeoff between Crane Work-load Balance and Ship Stability. *Proceedings of the …*, *III*, 1537–1543.

Imai, A., Sasaki, K., Nishimura, E., & Papadimitriou, S. (2006). Multi-objective simultaneous stowage and load planning for a *container* ship with *container* rehandle in yard stacks. *European Journal of Operational Research*, *171*(2), 373–389.

Lehnfeld, J., & Knust, S. (2014). Loading, unloading and premarshalling of stacks in storage areas: Survey and classification. *European Journal of Operational Research*, *239*(2), 297–312.

Martins, P. T., Lobo, V. J. a S., & Vairinhos, V. (2009). *Container* Stowage Problem Solution for Short Sea Shipping, (2001), 1–8.

Monaco, M. F., Sammarra, M., & Sorrentino, G. (2014). The terminal-oriented ship stowage planning problem. *European Journal of Operational Research*, *239*(1), 256–265.

Putamawa, F., & Santosa, B. (2011). *Pengembangan algoritma bee swarm optimization untuk penyelesaian container stowage problem*. Institut Teknologi Sepuluh Nopember. (Tugas Akhir).

Santosa, B., & Willy, P. (2011), *Metoda Metaheuristik Konsep dan Implementasi*, Surabaya, Guna Widya.

Wang, N., Jin, B., & Lim, A. (2015). Target-guided algorithms for the *container* pre-marshalling problem. *Omega (United Kingdom)*, *53*, 67–77.

Wang, S., Liu, Z., & Meng, Q. (2015). Segment-based alteration for *container* liner shipping network design. *Transportation Research Part B: Methodological*, *72*, 128–145.

Wilson, I. D., Roach, P. A., & Ware, J. A. (2001). *Container* stowage pre-planning: Using search to generate solutions, a case study. *Knowledge-Based Systems*, *14*(3-4), 137–145.

Zhang, R., Jin, Z., Ma, Y., & Luan, W. (2015). Optimization for two-stage double-cycle operations in *container* terminals. *Computers and Industrial Engineering*, *83*, 316–326.

LAMPIRAN A
MATLAB CODE

1. Data_konstrain

```
function
[Q,Q1,Q2,MT,MF]=Data_konstrain(C20ft,C40ft,tier,row,bay)
Q   = sum(C20ft(:,2))+sum(C40ft(:,2)); %(Total kapasitas
kapal)
Q1  = (Q/tier)/(row/2); %(Toleransi keseimbangan left-right)
Q2  = (Q/tier)/(bay/2); %(Toleransi keseimbangan anterior-
posterior)
MT  = sum(C20ft(:,2))/tier; %(Batas berat satu tingkat
container 20 feet)
MF  = sum(C40ft(:,2))/tier; %(Batas berat satu tingkat
container 40 feet
```

2. solusi_awal_loc.m (Pembangkitan kordinat iterasi 1)

```
function [R_A]=solusi_awal_loc(bay)
R_A=rand(1,fix(bay/2));
```

3. kordinat.m (Pembangkitan kordinat tiap N)

```
function
[Cordinat20,Cordinat40,R_co,A40]=kordinat(C40ft,bay,time,R_A)
[tier,row]=size(time);
JL20ft=bay*row;
JL40ft=fix(bay/2)*row;

[A40,~]=size(C40ft);
if A40<=JL40ft/2
    A_40=A40;
else
    A_40=JL40ft/2;
end

KL20ft=combvec(1:bay,1:row);
KL40ft=combvec(1:fix(bay/2),1:row);
x=ceil(A_40/row);
R_co=R_A;
[~,id]=sort(R_co,2);
id=id(1,1:x);
C40=[];
for i=1:size(id,2)
    for j=1:size(KL40ft,2)
        if KL40ft(1,j)==id(1,i)
            C40=[C40 KL40ft(:,j)];
        else
        end
    end
end
```

```
c20=[];
for i=id
    c20=[c20;i+(i-1) 2*i];
end

i=c20(:)';
for j=i
    [r,c]=find(KL20ft(1,:)==j);
    KL20ft(:,c)=0;
end
T=KL20ft~=0;
n=sum(T,2);
m=max(n);
B=nan(size(KL20ft,1),m);
for k=1:size(KL20ft,1)
    B(k,m-n(k)+1:m)=KL20ft(k,T(k,:));
end
Cordinat20=B;
Cordinat40=C40;
```

4. penyesuaian_matrik_i20.m (Penyamaan ukuran matrik tiap *tier* karena tiap *tier* banyaknya *container* tidak sama)

```
function [i_c20]=penyesuaian_matrik_i20(Cordinat20,i20)
[~,b]=size(Cordinat20);
[a,bb]=size(i20);
if i20==0
    i_c20=zeros(a,b);
else
    if b==bb,
        i_c20=i20;
    else
        i_c20=[i20 zeros(3,b-bb)];
    end
end
```

5. penyesuaian_matrik_i40.m

```
function [i_c40]=penyesuaian_matrik_i40(Cordinat40,i40)
[~,b]=size(Cordinat40);
[a,bb]=size(i40);
if i40==0
    i_c40=zeros(a,b);
else
    if b==bb,
        i_c40=i40;
    else
        i_c40=[i40 zeros(3,b-bb)];
    end
end
```

6. toleran_ant_post.m (Konstrain keseimbangan depan dan belakang)

```
function
[anterior,posterior]=toleran_ant_post(w20,w40,i20,i40,bay)
```

```
[~,J]=size(i20);
ant20=[];
post20=[];
ap20=[];
for r=1:J
    if mod(bay,2)==1
        if i20(2,r)==fix(bay/2)+1;
            ap20=[ap20;w20(r,2)];
        elseif i20(2,r)<=fix(bay/2);
            ant20=[ant20;w20(r,2)];
        else
            post20=[post20;w20(r,2)];
        end
    else
        if i20(2,r)<=fix(bay/2);
            ant20=[ant20;w20(r,2)];
        else
            post20=[post20;w20(r,2)];
        end
    end
end
t_ant20=sum(ant20(:))+(sum(ap20(:))/2);
t_post20=sum(post20(:))+(sum(ap20(:))/2);

[~,C]=size(i40);
ant40=[];
post40=[];
ap40=[];
bay40=fix(bay/2);
for r=1:C
    if mod(bay,2)==1
        if mod(bay40,2)==1
            if i40(2,r)==fix(bay40/2)+1
                ap40=[ap40;w40(r,2)];
            elseif i40(2,r)<=fix(bay40/2);
                ant40=[ant40;w40(r,2)];
            else
                post40=[post40;w40(r,2)];
            end
        else
            if i40(2,r)==(bay40/2)+1
                ap40=[ap40;w40(r,2)];
            elseif i40(2,r)<=(bay40/2);
                ant40=[ant40;w40(r,2)];
            else
                post40=[post40;w40(r,2)];
            end
        end
    else
        if mod(bay40,2)==1
            if i40(2,r)==fix(bay40/2)+1
                ap40=[ap40;w40(r,2)];
            elseif i40(2,r)<=fix(bay40/2);
                ant40=[ant40;w40(r,2)];
            else
                post40=[post40;w40(r,2)];
```

```
            end
        else
            if i40(2,r)<=(bay40/2);
                ant40=[ant40;w40(r,2)];
            else
                post40=[post40;w40(r,2)];
            end
        end
    end
end
tant40=sum(ant40(:));
tpost40=sum(post40(:));
tap40=sum(ap40(:));
if mod(bay,2)==1
    if mod(bay40,2)==1
        t_ant40=tant40+((tap40/4)*3);
        t_post40=tpost40+(tap40/4);
    else
        t_ant40=tant40+(tap40/4);
        t_post40=tpost40+((tap40/4)*3);
    end
else
    if mod(bay40,2)==1
        t_ant40=tant40+(tap40/2);
        t_post40=tpost40+(tap40/2);
    else
        t_ant40=tant40;
        t_post40=tpost40;
    end
end
anterior=t_ant20+t_ant40;
posterior=t_post20+t_post40;
end
```

7. toleran_left_right.m (Konstrain keseimbangan kanan dan kiri

```
function
[Tot_le,Tot_ri]=toleran_left_right(w20,w40,i20,i40,row)

[~,J]=size(i20);
le20=[];
ri20=[];
amidship20=[];
for r=1:J
    if mod(row,2)==1
        if i20(3,r)==1
            amidship20=[amidship20;w20(r,2)];
        elseif mod(i20(3,r),2)==0
            le20=[le20;w20(r,2)];
        else
            ri20=[ri20;w20(r,2)];
        end
    else
        if mod(i20(3,r),2)==0
            le20=[le20;w20(r,2)];
        else
            ri20=[ri20;w20(r,2)];
```

```
        end
    end
end
T_le20=sum(le20(:))+(sum(amidship20(:))/2);
T_ri20=sum(ri20(:))+(sum(amidship20(:))/2);

[~,C]=size(i40);
le40=[];
ri40=[];
amidship40=[];
for r=1:C
    if mod(row,2)==1
        if i40(3,r)==1
            amidship40=[amidship40;w40(r,2)];
        elseif mod(i40(3,r),2)==0
            le40=[le40;w40(r,2)];
        else
            ri40=[ri40;w40(r,2)];
        end

    else
        if mod(i40(3,r),2)==0
            le40=[le40;w40(r,2)];
        else
            ri40=[ri40;w40(r,2)];
        end
    end
end
T_le40=sum(le40(:))+(sum(amidship40(:))/2);
T_ri40=sum(ri40(:))+(sum(amidship40(:))/2);

Tot_le=T_le20+T_le40;
Tot_ri=T_ri20+T_ri40;
end
```

8. p_ur_con20.m (Menjamin kontainer yang lebih berat selalu dibawah, dan menghitung penalti tujuan yg lebih jauh berada dibawah tujuan yang lebih dekat)

```
function
[ic20,ii_c20,urut_c20,BTT20,kontrain_tujuan20]=p_ur_con20(ic20
,C20ft,tier,Cordinat20)
[~,d]=size(ic20);
xi_20=[];
for i=1:tier
    xi_20=[xi_20;((i-1)*d+1):(d*i)];
end
[aa,bb]=size(xi_20);
[a,b]=size(C20ft);
for i=1:aa
    for j=1:bb
        if xi_20(i,j)>a
            xi_20(i,j)=0;
        else
            xi_20(i,j)=xi_20(i,j);
```

```
        end
    end
end
for i=1:tier
    ic20((i-1)*3+1,:)=xi_20(i,:);
end
ii_c20=[];
p20=ic20;
for ti=1:tier
    pp=p20(((ti-1)*3)+1:3*ti,:);
    Cordin_20=[zeros(1,bb);Cordinat20];
    for ll=1:bb
        for jj=1:bb
            if Cordin_20(2:3,ll)==pp(2:3,jj)
                Cordin_20(1,ll)=pp(1,jj);
            end
        end
    end
    ii_c20=[ii_c20;Cordin_20];
end
uk_20=[];
for ti=1:tier
    dd=ii_c20((ti-1)*3+1,:);
    uk_20=[uk_20;dd];
end
[ga,ha]=size(uk_20);
for g=1:ga
    for h=1:ha
        if uk_20(g,h)==0
            B20(g,h)=0;
        else
            B20(g,h)=C20ft(uk_20(g,h),2);
        end
    end
end
BTT20=sum(B20,1);
[kk,kl]=sort(B20,1,'descend');
BZ_20=[];
for u=1:bb
    B=uk_20(kl(:,u),u);
    BZ_20=[BZ_20 B];
end
urut_c20=ii_c20;
for ti=1:tier
    urut_c20((ti-1)*3+1,:)=BZ_20(ti,:);
end

dest20=BZ_20;
[aj,sj]=size(dest20);
for i=1:aj
    for j=1:sj
        if dest20(i,j)==0
            dest20(i,j)=0;
        else
            dest20(i,j)=C20ft(dest20(i,j),3);
        end
    end
```

```
end
if tier>=1
    kontrain_tujuan20=dest20;
else
    z20=[];
    for i=1:tier-1
        x=zeros(1,bb);
        for j=1:bb
            if dest20(i,j)>=dest20(i+1,j)
                x(1,j)=0;
            else
                x(1,j)=1;
            end
        end
        z20=[z20;x];
    end
    zz20=sum(z20,1);
    for zi=1:bb
        if zz20(1,zi)>=1
            zz20(1,zi)=1;
        else
            zz20(1,zi)=0;
        end
    end
    kontrain_tujuan20=sum(zz20(:));
end
```

9. p_ur_con40.m

```
function
[ic40,ii_c40,urut_c40,BTT40,kontrain_tujuan40]=p_ur_con40(ic40
,C40ft,tier,Cordinat40)
[~,d]=size(ic40);
xi_40=[];
for i=1:tier
    xi_40=[xi_40;((i-1)*d+1):(d*i)];
end
[aa,bb]=size(xi_40);
[a,b]=size(C40ft);
for i=1:aa
    for j=1:bb
        if xi_40(i,j)>a
            xi_40(i,j)=0;
        else
            xi_40(i,j)=xi_40(i,j);
        end
    end
end
for i=1:tier
    ic40((i-1)*3+1,:)=xi_40(i,:);
end
ii_c40=[];
p40=ic40;
for ti=1:tier
    pp=p40(((ti-1)*3)+1:3*ti,:);
```

```
    Cordin_40=[zeros(1,bb);Cordinat40];
    for ll=1:bb
        for jj=1:bb
            if Cordin_40(2:3,ll)==pp(2:3,jj)
                Cordin_40(1,ll)=pp(1,jj);
            end
        end
    end
    ii_c40=[ii_c40;Cordin_40];
end
uk_40=[];
for ti=1:tier
    dd=ii_c40((ti-1)*3+1,:);
    uk_40=[uk_40;dd];
end
[ga,ha]=size(uk_40);
for g=1:ga
    for h=1:ha
        if uk_40(g,h)==0
            B40(g,h)=0;
        else
            B40(g,h)=C40ft(uk_40(g,h),2);
        end
    end
end
BTT40=sum(B40,1);
[kk,kl]=sort(B40,1,'descend');
BZ_40=[];
for u=1:bb
    B=uk_40(kl(:,u),u);
    BZ_40=[BZ_40 B];
end
urut_c40=ii_c40;
for ti=1:tier
    urut_c40((ti-1)*3+1,:)=BZ_40(ti,:);
end

dest40=BZ_40;
[aj,sj]=size(dest40);
for i=1:aj
    for j=1:sj
        if dest40(i,j)==0
            dest40(i,j)=0;
        else
            dest40(i,j)=C40ft(dest40(i,j),3);
        end
    end
end
if tier>=1
    kontrain_tujuan40=dest40;
else
    z40=[];
    for i=1:tier-1
        x=zeros(1,bb);
        for j=1:bb
            if dest40(i,j)>=dest40(i+1,j)
                x(1,j)=0;
```

```
            else
                x(1,j)=1;
            end
        end
        z40=[z40;x];
    end
    zz40=sum(z40,1);
    for zi=1:bb
        if zz40(1,zi)>=1
            zz40(1,zi)=1;
        else
            zz40(1,zi)=0;
        end
    end
    kontrain_tujuan40=sum(zz40(:));
end
```

10. CSP_APSO.m

```
function
[R_co,N_40,N_20,fit,tier]=CSP_APSO(C20ft,C40ft,bay,time,R_A)
[Q,Q1,Q2,MT,MF]=Data_konstrain;
[tier,row]=size(time);
[Cordinat20,Cordinat40,R_co,A40]=kordinat(C40ft,bay,time,R_A);
Q2_le=[];
Q2_ri=[];
Q1_ant=[];
Q1_post=[];
N_40=[];
N_20=[];
ic20=[];
ic40=[];
for ti=1:tier
    if ti==1
        [~,K]=size(Cordinat40);
        m_40=A40-(K*(ti-1));
        if m_40<=K
            m401=m_40;
        else
            m401=K;
        end

        [~,J]=size(Cordinat20);
        [A20,~]=size(C20ft);
        m_20=A20-(J*(ti-1));
        if m_20<=J
            m201=m_20;
        else
            m201=J;
        end

        Z40ft=size(Cordinat40);
        N40=rand(1,Z40ft(1,2));
        [rand40 idk40]=sort(N40,2);
        c40=[idk40;Cordinat40];
        [Y,I]=sort(c40(1,:));
        c40=c40(:,I);
```

```
        i40=c40(:,1:m401);

        Z20ft=size(Cordinat20);
        N20=rand(1,Z20ft(1,2));
        [rand20 idk20]=sort(N20,2);
        c20=[idk20;Cordinat20];
        [Y,I]=sort(c20(1,:));
        c20=c20(:,I);
        i20=c20(:,1:m201);
        [i_c20]=penyesuaian_matrik_i20(Cordinat20,i20);
        [i_c40]=penyesuaian_matrik_i40(Cordinat40,i40);

        w20=C20ft(1:m201,:);
        w40=C40ft(1:m401,:);

[Tot_le,Tot_ri]=toleran_left_right(w20,w40,i20,i40,row);
        Q2_le=[Q2_le;Tot_le];
        Q2_ri=[Q2_ri;Tot_ri];

[anterior,posterior]=toleran_ant_post(w20,w40,i20,i40,bay);
        Q1_ant=[Q1_ant;anterior];
        Q1_post=[Q1_post;posterior];


fitness(ti,:)=sum(time(ti,i20(3,:)))+sum(time(ti,i40(3,:)));

    else

        [~,K]=size(Cordinat40);
        m_40=A40-(K*(ti-1));
        if m_40<=K
            m40=m_40;
        else
            m40=K;
        end

        [~,J]=size(Cordinat20);
        [A20,~]=size(C20ft);
        m_20=A20-(J*(ti-1));
        if m_20<=J
            m20=m_20;
        else
            m20=J;
        end

        Z40ft=size(Cordinat40);
        N40=rand(1,Z40ft(1,2));
        [rand40 idk40]=sort(N40,2);
        c40=[idk40;Cordinat40];
        [Y,I]=sort(c40(1,:));
        c40=c40(:,I);
        i40=c40(:,1:m40);

        Z20ft=size(Cordinat20);
        N20=rand(1,Z20ft(1,2));
        [rand20 idk20]=sort(N20,2);
```

```
        c20=[idk20;Cordinat20];
        [Y,I]=sort(c20(1,:));
        c20=c20(:,I);
        i20=c20(:,1:m20);
        [i_c20]=penyesuaian_matrik_i20(Cordinat20,i20);
        [i_c40]=penyesuaian_matrik_i40(Cordinat40,i40);

        w20=C20ft(((m201*ti)+1)-m201:((m201*ti)-m201)+m20,:);
        w40=C40ft(((m401*ti)+1)-m401:((m401*ti)-m401)+m40,:);

[Tot_le,Tot_ri]=toleran_left_right(w20,w40,i20,i40,row);
        Q2_le=[Q2_le;Tot_le];
        Q2_ri=[Q2_ri;Tot_ri];

[anterior,posterior]=toleran_ant_post(w20,w40,i20,i40,bay);
        Q1_ant=[Q1_ant;anterior];
        Q1_post=[Q1_post;posterior];


fitness(ti,:)=sum(time(ti,i20(3,:)))+sum(time(ti,i40(3,:)));

    end
    ic20=[ic20;i_c20];
    ic40=[ic40;i_c40];

    Q2le=sum(Q2_le(:));
    Q2ri=sum(Q2_ri(:));
    if -Q1<=(Q1_ant-Q1_post) & (Q1_ant-Q1_post)<=Q1 & -
Q2<=(Q2le-Q2ri) & (Q2le-Q2ri)<=Q2
        k2=0;
    else
        k2=100;
    end
    fit=sum(fitness(:))+k2;

    N_40=[N_40;N40];
    N_20=[N_20;N20];
end
[ic20,ii_c20,urut_c20,BTT20,kontrain_tujuan20]=p_ur_con20(ic20
,C20ft,tier,Cordinat20);
[ic40,ii_c40,urut_c40,BTT40,kontrain_tujuan40]=p_ur_con40(ic40
,C40ft,tier,Cordinat40);
s20=C20ft(:,2);
s40=C40ft(:,2);
SQ=sum(s20(:))+sum(s40(:));
if BTT20(:,:)<=MT
    k01=0;
else
    k01=1;
end
if BTT40(:,:)<=MF
    k11=0;
else
    k11=1;
end
if SQ<=Q
    k21=0;
```

```
else
    k21=1;
end
if k01+k11+k21==0
    k1=0;
else
    k1=1000000;
end
fit=fit+k1;
end
```

11. CSP_APSO_update.m (Perhitungan update)

```
function
[R_co,N_40,N_20,fit,tier]=CSP_APSO_update(C20ft,C40ft,bay,time
,rand_N40,rand_N20,rand_cordinat)
[Q,Q1,Q2,MT,MF]=Data_konstrain;
[tier,row]=size(time);
[Cordinat20,Cordinat40,R_co,A40]=kordinat(C40ft,bay,time,rand_
cordinat);
Q2_le=[];
Q2_ri=[];
Q1_ant=[];
Q1_post=[];
N_40=[];
N_20=[];
ic20=[];
ic40=[];
for ti=1:tier
    if ti==1
        [~,K]=size(Cordinat40);
        m_40=A40-(K*(ti-1));
        if m_40<=K
            m401=m_40;
        else
            m401=K;
        end

        [~,J]=size(Cordinat20);
        [A20,~]=size(C20ft);
        m_20=A20-(J*(ti-1));
        if m_20<=J
            m201=m_20;
        else
            m201=J;
        end

        Z40ft=size(Cordinat40);
        N40=rand_N40(ti,:);
        [rand40 idk40]=sort(N40,2);
        c40=[idk40;Cordinat40];
        [Y,I]=sort(c40(1,:));
        c40=c40(:,I);
        i40=c40(:,1:m401);

        Z20ft=size(Cordinat20);
```

```
        N20=rand_N20(ti,:);
        [rand20 idk20]=sort(N20,2);
        c20=[idk20;Cordinat20];
        [Y,I]=sort(c20(1,:));
        c20=c20(:,I);
        i20=c20(:,1:m201);
        [i_c20]=penyesuaian_matrik_i20(Cordinat20,i20);
        [i_c40]=penyesuaian_matrik_i40(Cordinat40,i40);

        w20=C20ft(1:m201,:);
        w40=C40ft(1:m401,:);

[Tot_le,Tot_ri]=toleran_left_right(w20,w40,i20,i40,row);
        Q2_le=[Q2_le;Tot_le];
        Q2_ri=[Q2_ri;Tot_ri];

[anterior,posterior]=toleran_ant_post(w20,w40,i20,i40,bay);
        Q1_ant=[Q1_ant;anterior];
        Q1_post=[Q1_post;posterior];


fitness(ti,:)=sum(time(ti,i20(3,:)))+sum(time(ti,i40(3,:)));

    else

        [~,K]=size(Cordinat40);
        m_40=A40-(K*(ti-1));
        if m_40<=K
            m40=m_40;
        else
            m40=K;
        end

        [~,J]=size(Cordinat20);
        [A20,~]=size(C20ft);
        m_20=A20-(J*(ti-1));
        if m_20<=J
            m20=m_20;
        else
            m20=J;
        end

        Z40ft=size(Cordinat40);
        N40=rand_N40(ti,:);
        [rand40 idk40]=sort(N40,2);
        c40=[idk40;Cordinat40];
        [Y,I]=sort(c40(1,:));
        c40=c40(:,I);
        i40=c40(:,1:m40);

        Z20ft=size(Cordinat20);
        N20=rand_N20(ti,:);
        [rand20 idk20]=sort(N20,2);
        c20=[idk20;Cordinat20];
        [Y,I]=sort(c20(1,:));
        c20=c20(:,I);
```

```
        i20=c20(:,1:m20);
        [i_c20]=penyesuaian_matrik_i20(Cordinat20,i20);
        [i_c40]=penyesuaian_matrik_i40(Cordinat40,i40);

        w20=C20ft(((m201*ti)+1)-m201:((m201*ti)-m201)+m20,:);
        w40=C40ft(((m401*ti)+1)-m401:((m401*ti)-m401)+m40,:);

[Tot_le,Tot_ri]=toleran_left_right(w20,w40,i20,i40,row);
        Q2_le=[Q2_le;Tot_le];
        Q2_ri=[Q2_ri;Tot_ri];

[anterior,posterior]=toleran_ant_post(w20,w40,i20,i40,bay);
        Q1_ant=[Q1_ant;anterior];
        Q1_post=[Q1_post;posterior];


fitness(ti,:)=sum(time(ti,i20(3,:)))+sum(time(ti,i40(3,:)));

    end
    ic20=[ic20;i_c20];
    ic40=[ic40;i_c40];

    Q2le=sum(Q2_le(:));
    Q2ri=sum(Q2_ri(:));
    if -Q1<=(Q1_ant-Q1_post) & (Q1_ant-Q1_post)<=Q1 & -
Q2<=(Q2le-Q2ri) & (Q2le-Q2ri)<=Q2
        k2=0;
    else
        k2=100;
    end
    fit=sum(fitness(:))+k2;

    N_40=[N_40;N40];
    N_20=[N_20;N20];
end
[ic20,ii_c20,urut_c20,BTT20,kontrain_tujuan20]=p_ur_con20(ic20
,C20ft,tier,Cordinat20);
[ic40,ii_c40,urut_c40,BTT40,kontrain_tujuan40]=p_ur_con40(ic40
,C40ft,tier,Cordinat40);
s20=C20ft(:,2);
s40=C40ft(:,2);
SQ=sum(s20(:))+sum(s40(:));
if BTT20(:,:)<=MT
    k01=0;
else
    k01=1;
end
if BTT40(:,:)<=MF
    k11=0;
else
    k11=1;
end
if SQ<=Q
    k21=0;
else
    k21=1;
end
```

```
if k01+k11+k21==0
    k1=0;
else
    k1=1000000;
end
fit=fit+k1;
end
```

12. CSP_menggunakan_PSO.m

```
function
[pbest_R,pbest_N40,pbest_N20,R_co1b,R_N40,R_N20,f,f_b,pbestR,p
bestN40,pbestN20,it]=CSP_menggunakan_PSO(C20ft,C40ft,time,bay,
N)
R_co1=[];
N_401=[];
N_201=[];
f=[];
for i=1:N

[R_co,N_40,N_20,fit,tier]=CSP_APSO(C20ft,C40ft,bay,time,solusi
_awal_loc(bay));
    f=[f;fit];
    R_co1=[R_co1;R_co];
    N_401=[N_401;N_40];
    N_201=[N_201;N_20];
end
[~,idk_f]=min(f);

gbest_R=R_co1(idk_f,:);
gbest_N40=N_401(((idk_f*tier)-(tier-1)):(idk_f*tier),:);
gbest_N20=N_201(((idk_f*tier)-(tier-1)):(idk_f*tier),:);

pbest_R=R_co1;
pbest_N40=N_401;
pbest_N20=N_201;



[~,a]=size(R_co1);
[H,b]=size(gbest_N40);
[I,c]=size(gbest_N20);
v_R_co1=zeros(N,a);
v_pbest40=zeros(N*H,b);
v_pbest20=zeros(N*I,c);
it=1;
max_it=10;
while it<=max_it
    w(it)=0.9-(0.5/max_it)*it;
    v_R_co1=w(it)*v_R_co1+rand(N,a).*(pbest_R-
R_co1)+rand(N,a).*(repmat(gbest_R,N,1)-R_co1);
    R_co1b=v_R_co1+R_co1;
    [L,G]=size(R_co1b);
    for i=1:L
```

```
        for j=1:G
            if R_co1b(i,j)>1
                R_co1b(i,j)=1;
            elseif R_co1b(i,j)<-1
                R_co1b(i,j)=-1;
            end
        end
    end
    v_N40=w(it)*v_pbest40+rand(N*H,b).*(pbest_N40-
N_401)+rand(N*H,b).*(repmat(gbest_N40,N,1)-N_401);
    R_N40=v_N40+N_401;
    [U,T]=size(R_N40);
    for v=1:U
        for o=1:T
            if R_N40(v,o)>1
                R_N40(v,o)=1;
            elseif R_N40(v,o)<-1
                R_N40(v,o)=-1;
            end
        end
    end
    v_N20=w(it)*v_pbest20+rand(N*I,c).*(pbest_N20-
N_201)+rand(N*I,c).*(repmat(gbest_N20,N,1)-N_201);
    R_N20=v_N20+N_201;
    [K,L]=size(R_N20);
    for dd=1:K
        for oo=1:L
            if R_N20(dd,oo)>1
                R_N20(dd,oo)=1;
            elseif R_N20(dd,oo)<-1
                R_N20(dd,oo)=-1;
            end
        end
    end

    R_co1_b=[];
    N_401_b=[];
    N_201_b=[];
    f_b=[];
    for ii=1:N
        rand_cordinat=R_co1b(ii,:);
        rand_N40=R_N40(((ii*tier)-(tier-1)):(ii*tier),:);
        rand_N20=R_N20(((ii*tier)-(tier-1)):(ii*tier),:);


[R_co,N_40,N_20,fit,tier]=CSP_APSO_update(C20ft,C40ft,bay,time
,rand_N40,rand_N20,rand_cordinat);

        f_b=[f_b;fit];
        R_co1_b=[R_co1_b;R_co];
        N_401_b=[N_401_b;N_40];
        N_201_b=[N_201_b;N_20];
    end

    [ar,br]=size(R_co1b);
    [a4,b4]=size(R_N40);
    [a2,b2]=size(R_N20);
```

```
    pbestR=zeros(ar,br);
    pbestN40=zeros(a4,b4);
    pbestN20=zeros(a2,b2);

    for i=1:N
        if f(i,1)<=f_b(i,1)
            pbestR(i,:)=pbest_R(i,:);
            pbestN40(((i*tier)-(tier-
1)):(i*tier),:)=pbest_N40(((i*tier)-(tier-1)):(i*tier),:);
            pbestN20(((i*tier)-(tier-
1)):(i*tier),:)=pbest_N20(((i*tier)-(tier-1)):(i*tier),:);
        else
            pbestR(i,:)=R_co1b(i,:);
            pbestN40(((i*tier)-(tier-
1)):(i*tier),:)=R_N40(((i*tier)-(tier-1)):(i*tier),:);
            pbestN20(((i*tier)-(tier-
1)):(i*tier),:)=R_N20(((i*tier)-(tier-1)):(i*tier),:);
        end
    end
    fbest=zeros(N,1);
    for i=1:N
        if f(i,1)<=f_b(i,1)
            fbest(i,1)=f(i,1);
        else
            fbest(i,1)=f_b(i,1);
        end
    end
    [~,idk_fbest]=min(fbest);
    gbest_R=pbestR(idk_fbest,:);
    gbest_N40=pbestN40(((idk_fbest*tier)-(tier-
1)):(idk_fbest*tier),:);
    gbest_N20=pbestN20(((idk_fbest*tier)-(tier-
1)):(idk_fbest*tier),:);

    pbest_R=pbestR;
    R_co1=R_co1_b;

    pbest_N40=pbestN40;
    N_401=N_401_b;

    pbest_N20=pbestN20;
    N_201=N_201_b;


    it=it+1;

end
```

13. Data_konstrain2.m

```
function
[Q,Q1,Q2,MT,MF]=Data_konstrain2(C20ft,C40ft,tier,row,bay)
Q   = sum(C20ft(:,2))+sum(C40ft(:,2)); %(Total kapasitas
kapal)
Q1  = Q/(row/2); %(Toleransi keseimbangan left-right)
Q2  = Q/(bay/2); %(Toleransi keseimbangan anterior-posterior)
```

```
MT  = sum(C20ft(:,2))/tier; %(Batas berat satu tingkat
container 20 feet)
MF  = sum(C40ft(:,2))/tier; %(Batas berat satu tingkat
container 40 feet
```

14. rubah_tumpukan20.m

```
function
[wZ_20,tier,row,cor_20,w3,tu20,w20,i20]=rubah_tumpukan20(P_C20
ft,C20ft,time)
[tier,row]=size(time);
[~,yi]=size(P_C20ft);
cor_20=P_C20ft(2:3,:);
y=zeros(tier,yi);
for t=1:tier
    y(t,:)=P_C20ft((t-1)*3+1,:);
end
%=========
ze=find(y==0);
[i_ze,~]=size(ze);
s=fix(i_ze/tier);
%=========
for st=1:s

    [r,c]=size(y);
    z=zeros(r,c);
    for i=1:r
        for j=1:c
            if y(i,j)==0
                z(i,j)=1;
            end
        end
    end
    zz=sum(z);
    [~,d]=max(zz);
    k=y(:,d);
    k_f=find(k>0);
    [b,~]=size(k_f);

    for ti=1:b
        [r,c]=size(y);
        z=zeros(r,c);
        for i=1:r
            for j=1:c
                if y(i,j)==0
                    z(i,j)=1;
                end
            end
        end
        zz=sum(z);
        [~,d]=max(zz);
        k=y(:,d);
        k_f=find(k>0);
        [b,~]=size(k_f);


        k_n=k(b,1);
        y(b,d)=0;
```

```
        y_r=y(:,d);
        y(:,d)=1000;

        [ir,ic]=find(y==0);
        mb=[ir ic];
        [i_mb,~]=size(mb);
        mb=[mb;zeros(c-i_mb,2)];
        [i_mb,~]=size(mb);
        [i_k_n,~]=size(k_n);
        ik=[k_n;zeros(i_mb-i_k_n,1)];
        m_b=[mb ik];

        for i=1:r
            for j=1:c
                if m_b(j,1) & m_b(j,2)
                    y(m_b(j,1),m_b(j,2))=m_b(j,3);
                end
            end
        end
        y(:,d)=y_r;
    end
    y(:,d)=100;
end
[rr,cc]=size(y);
for i=1:rr
    for j=1:cc
        if y(i,j)==100
            y(i,j)=0;
        end
    end
end
%urutan tumpukan berdasarkan tujuan
B20=zeros(rr,cc);
w20=zeros(rr,cc);
for g=1:rr
    for h=1:cc
        if y(g,h)==0
            B20(g,h)=0;
            w20(g,h)=0;
        else
            B20(g,h)=C20ft(y(g,h),3);
            w20(g,h)=C20ft(y(g,h),2);
        end
    end
end
[kk,kl]=sort(B20,1,'descend');
BZ_20=[];
for u=1:cc
    B=y(kl(:,u),u);
    BZ_20=[BZ_20 B];
end
%urutan tumpukan berdasarkan berat light (5-15 ton),medium
(15-25 ton),
%dan heavy (lebih dari 25 ton)
y=BZ_20;
w3=zeros(rr,cc);
for wr=1:rr
```

```
    for wc=1:cc
        if y(wr,wc)==0
            w3(wr,wc)=0;
        elseif y(wr,wc)>=1 & y(wr,wc)<=15
            w3(wr,wc)=15;
        elseif y(wr,wc)>=15 & y(wr,wc)<=25
            w3(wr,wc)=25;
        else
            w3(wr,wc)=35;
        end
    end
end
[wk,wl]=sort(w3,1,'descend');
wZ_20=[];
for u=1:cc
    w=y(wl(:,u),u);
    wZ_20=[wZ_20 w];
end
%bentuk output i20 dan w20
cy=[];
ur=[];
y= wZ_20;
for i=1:tier
    ucy=y(i,:);
    u=cor_20;
    cy=[cy ucy];
    ur=[ur u];
    i20=[cy;ur];
end

[~,N_i20]=size(i20);
i2=[];
for j=1:N_i20
    if i20(1,j)>0
        Ni2=i20(:,j);
        i2=[i2 Ni2];
    end
end
i20=i2;

w_20=i20';
wb_20=[];
for k=w_20(:,1)
    N_wb20=C20ft(k,2);
    wb_20=[wb_20;N_wb20];
end
w20=[w_20(:,1) wb_20];

%perhitungan fitness
c20=wZ_20;
[a,b]=size(c20);
unloading20=zeros(a,b);
cordinat20=P_C20ft(3,:);
for i=1:a
    for j=1:b
        if c20(i,j)>0
            unloading20(i,j)=time(i,cordinat20(1,j));
```

```
        end
    end
end
tu20=sum(unloading20(:));
end
```

15. rubah_tumpukan40.m

```
function
[wZ_40,cor_40,w4,tu40,w40,i40]=rubah_tumpukan40(P_C40ft,C40ft,
time)
[tier,row]=size(time);
[~,yi]=size(P_C40ft);
cor_40=P_C40ft(2:3,:);
y=zeros(tier,yi);
for t=1:tier
    y(t,:)=P_C40ft((t-1)*3+1,:);
end
%=========
ze=find(y==0);
[i_ze,~]=size(ze);
s=fix(i_ze/tier);
%=========
for st=1:s

    [r,c]=size(y);
    z=zeros(r,c);
    for i=1:r
        for j=1:c
            if y(i,j)==0
                z(i,j)=1;
            end
        end
    end
    zz=sum(z);
    [~,d]=max(zz);
    k=y(:,d);
    k_f=find(k>0);
    [b,~]=size(k_f);

    for ti=1:b
        [r,c]=size(y);
        z=zeros(r,c);
        for i=1:r
            for j=1:c
                if y(i,j)==0
                    z(i,j)=1;
                end
            end
        end
        zz=sum(z);
        [~,d]=max(zz);
        k=y(:,d);
        k_f=find(k>0);
        [b,~]=size(k_f);


        k_n=k(b,1);
```

```
        y(b,d)=0;
        y_r=y(:,d);
        y(:,d)=1000;

        [ir,ic]=find(y==0);
        mb=[ir ic];
        [i_mb,~]=size(mb);
        mb=[mb;zeros(c-i_mb,2)];
        [i_mb,~]=size(mb);
        [i_k_n,~]=size(k_n);
        ik=[k_n;zeros(i_mb-i_k_n,1)];
        m_b=[mb ik];

        for i=1:r
            for j=1:c
                if m_b(j,1) & m_b(j,2)
                    y(m_b(j,1),m_b(j,2))=m_b(j,3);
                end
            end
        end
        y(:,d)=y_r;
    end
    y(:,d)=100;
end
[rr,cc]=size(y);
for i=1:rr
    for j=1:cc
        if y(i,j)==100
            y(i,j)=0;
        end
    end
end
%urutan tumpukan berdasarkan tujuan
B40=zeros(rr,cc);
w40=zeros(rr,cc);
for g=1:rr
    for h=1:cc
        if y(g,h)==0
            B40(g,h)=0;
            w40(g,h)=0;
        else
            B40(g,h)=C40ft(y(g,h),3);
            w40(g,h)=C40ft(y(g,h),2);
        end
    end
end
[kk,kl]=sort(B40,1,'descend');
BZ_40=[];
for u=1:cc
    B=y(kl(:,u),u);
    BZ_40=[BZ_40 B];
end
%urutan tumpukan berdasarkan berat light (5-15 ton),medium
(15-25 ton),
%dan heavy (lebih dari 25 ton)
y=BZ_40;
w4=zeros(rr,cc);
```

```
for wr=1:rr
    for wc=1:cc
        if y(wr,wc)==0
            w4(wr,wc)=0;
        elseif y(wr,wc)>=1 & y(wr,wc)<=15
            w4(wr,wc)=15;
        elseif y(wr,wc)>=15 & y(wr,wc)<=25
            w4(wr,wc)=25;
        else
            w4(wr,wc)=35;
        end
    end
end
[wk,wl]=sort(w4,1,'descend');
wZ_40=[];
for u=1:cc
    w=y(wl(:,u),u);
    wZ_40=[wZ_40 w];
end
%bentuk output i40 dan w40
cy=[];
ur=[];
y= wZ_40;
for i=1:tier
    ucy=y(i,:);
    u=cor_40;
    cy=[cy ucy];
    ur=[ur u];
    i40=[cy;ur];
end

[~,N_i40]=size(i40);
i4=[];
for j=1:N_i40
    if i40(1,j)>0
        Ni4=i40(:,j);
        i4=[i4 Ni4];
    end
end
i40=i4;

w_40=i40';
wb_40=[];
for k=w_40(:,1)
    N_wb40=C40ft(k,2);
    wb_40=[wb_40;N_wb40];
end
w40=[w_40(:,1) wb_40];

%perhitungan fitness
c40=wZ_40;
[a,b]=size(c40);
unloading40=zeros(a,b);
cordinat40=P_C40ft(3,:);
for i=1:a
    for j=1:b
        if c40(i,j)>0
```

```
            unloading40(i,j)=time(i,cordinat40(1,j));
        end
    end
end
tu40=sum(unloading40(:));
end
```

16. PSO_CSP_MODIF.m

```
function
[it,kordinat_C20,PC20,kordinat_C40,PC40,f_b,f_bb,fitness,start
_time]=PSO_CSP_MODIF(C20ft,C40ft,bay,time,N)
start_time=cputime;
[tier,~]=size(time);
[posisi_C20ft,posisi_C40ft,P_C20ft,P_C40ft,f_b,it]=CSP_menggun
akan_PSO(C20ft,C40ft,time,bay,N);
kordinat_C20=P_C20ft(2:3,:);
kordinat_C40=P_C40ft(2:3,:);

fitness=[];
PC20=[];
PC40=[];
f_bb=[];
for i=1:N
    P_C20ft=posisi_C20ft((i*3*tier+1)-3*tier:i*3*tier,:);
    P_C40ft=posisi_C40ft((i*3*tier+1)-3*tier:i*3*tier,:);

    PC_20_L=[];
    PC_40_L=[];
    for t=1:tier
        y20=P_C20ft((t-1)*3+1,:);
        y40=P_C40ft((t-1)*3+1,:);
        PC_20_L=[PC_20_L;y20];
        PC_40_L=[PC_40_L;y40];
    end
    %=================

[wZ_20,tier,row,cor_20,w3,tu20,w20,i20]=rubah_tumpukan20(P_C20
ft,C20ft,time);

[wZ_40,cor_40,w4,tu40,w40,i40]=rubah_tumpukan40(P_C40ft,C40ft,
time);


[anterior,posterior]=toleran_ant_post(w20,w40,i20,i40,bay);
    [Tot_le,Tot_ri]=toleran_left_right(w20,w40,i20,i40,row);

    [Q,Q1,Q2,MT,MF]=Data_konstrain2(C20ft,C40ft,tier,row,bay);
    M20=sum(w3);
    M40=sum(w4);
    TM=sum(w20(:,2))+sum(w40(:,2));
    if M20(:,:)<=MT
        k01=0;
    else
        k01=1;
    end
    if M40(:,:)<=MF
        k11=0;
```

```
    else
        k11=1;
    end
    if TM<=Q
        k21=0;
    else
        k21=1;
    end
    if k01+k11+k21==0
        k1=0;
    else
        k1=1000000;
    end
    if -Q1<=(anterior-posterior) & (anterior-posterior)<=Q1 &
-Q2<=(Tot_le-Tot_ri) & (Tot_le-Tot_ri)<=Q2
        k2=0;
    else
        k2=100;
    end
    pinalty=k1+k2;
    fitnes=tu20+tu40+pinalty;
    f_bb=[f_bb;fitnes];

    if fitnes<f_b(N,1)
        fit=fitnes;
        P20=wZ_20;
        P40=wZ_40;
    else
        fit=f_b(N,1);
        P20=PC_20_L;
        P40=PC_40_L;
    end

    fitness=[fitness;fit];
    PC20=[PC20;P20];
    PC40=[PC40;P40];
    %============================
end

for l=1:N
    PC_20=PC20((l-1)*tier+1:l*tier,:);
    PC_40=PC40((l-1)*tier+1:l*tier,:);

disp('N=');disp(l);disp('PC_20');disp(PC_20);disp('PC_40');dis
p(PC_40);
end
```

(Halaman ini sengaja dikosongkan)

LAMPIRAN B
DATA

B.1 Kasus 1

Kontainer ukuran 20								
No	Berat	Tujuan	No	Berat	Tujuan	No	Berat	Tujuan
1	20	2	22	10	1	43	15	2
2	35	1	23	10	2	44	25	2
3	15	1	24	10	2	45	20	2
4	35	1	25	10	1	46	30	2
5	25	1	26	10	2	47	25	1
6	35	2	27	30	2	48	20	2
7	25	2	28	10	1	49	35	1
8	25	2	29	10	1	50	30	2
9	30	1	30	10	1	51	20	1
10	20	2	31	15	1	52	30	2
11	20	1	32	15	1	53	15	2
12	15	2	33	15	2	54	25	2
13	15	2	34	15	2	55	30	2
14	25	1	35	10	2	56	35	1
15	10	2	36	10	1	57	30	1
16	10	1	37	10	1	58	35	1
17	35	1	38	25	2	59	35	1
18	15	1	39	35	2	60	10	2
19	25	2	40	25	2	61	25	1
20	15	1	41	20	2	62	10	1
21	30	1	42	20	2			

Kontainer ukuran 40								
No	Berat	Tujuan	No	Berat	Tujuan	No	Berat	Tujuan
1	15	2	14	10	2	27	25	1
2	20	2	15	25	1	28	15	2
3	10	2	16	10	2	29	10	2
4	35	1	17	20	1	30	30	2
5	15	1	18	10	2	31	20	2
6	30	2	19	10	2	32	35	1
7	10	1	20	15	1	33	25	1
8	15	1	21	35	2	34	15	2
9	15	1	22	20	2	35	30	1
10	15	2	23	25	1	36	25	1
11	35	2	24	10	2	37	15	2
12	10	2	25	30	1	38	25	2
13	15	1	26	25	1			

B.2 Kasus 2

Kontainer ukuran 20								
No	Berat	Tujuan	No	Berat	Tujuan	No	Berat	Tujuan
1	10	2	26	10	2	51	20	2
2	30	2	27	15	2	52	25	2
3	20	1	28	25	1	53	10	1
4	30	1	29	30	2	54	10	2
5	30	2	30	10	2	55	35	2
6	15	2	31	15	2	56	10	1
7	15	1	32	10	1	57	15	2
8	10	1	33	20	2	58	35	2
9	25	2	34	10	2	59	20	2
10	15	2	35	30	1	60	15	1
11	20	2	36	25	2	61	25	1
12	10	1	37	15	2	62	25	2
13	30	2	38	35	1	63	30	2
14	25	2	39	30	1	64	20	2
15	10	2	40	10	1	65	20	1
16	15	1	41	25	1	66	20	2
17	10	1	42	20	1	67	20	1
18	15	1	43	25	1	68	35	2
19	10	2	44	15	1	69	25	1
20	20	1	45	10	2	70	10	1
21	15	2	46	15	2	71	20	1
22	15	2	47	15	2	72	30	1
23	35	1	48	25	1	73	15	1
24	20	1	49	35	1	74	15	1
25	15	2	50	20	1	75	20	1

Kontainer ukuran 40								
No	Berat	Tujuan	No	Berat	Tujuan	No	Berat	Tujuan
1	30	2	16	20	2	31	30	1
2	25	2	17	10	1	32	20	2
3	10	2	18	15	1	33	25	1
4	10	2	19	10	2	34	15	2
5	10	2	20	20	2	35	10	2
6	25	1	21	25	1	36	25	2
7	30	2	22	20	1	37	15	1
8	10	1	23	35	1	38	30	1
9	20	1	24	35	1	39	35	2
10	30	2	25	15	2	40	30	2
11	20	1	26	25	1	41	20	2
12	25	1	27	25	1	42	25	2
13	10	2	28	35	2	43	20	2
14	20	2	29	15	2	44	15	2
15	10	2	30	35	2	45	15	1

B.3 Kasus 3

Kontainer ukuran 20								
No	Berat	Tujuan	No	Berat	Tujuan	No	Berat	Tujuan
1	15	2	31	15	1	61	35	1
2	15	2	32	10	1	62	25	1
3	10	1	33	30	2	63	20	2
4	10	1	34	10	1	64	10	1
5	20	1	35	15	2	65	15	2
6	25	1	36	20	1	66	20	2
7	25	1	37	10	2	67	15	1
8	10	2	38	15	2	68	25	2
9	15	2	39	10	2	69	15	1
10	20	1	40	35	1	70	25	1
11	35	2	41	20	2	71	25	2
12	10	1	42	30	1	72	20	2
13	15	2	43	10	1	73	10	1
14	25	2	44	25	2	74	35	2
15	30	2	45	35	2	75	30	1
16	20	2	46	35	2	76	30	1
17	15	1	47	20	1	77	35	1
18	15	1	48	10	2	78	35	1
19	15	1	49	20	2	79	25	2
20	15	1	50	20	1	80	15	2
21	15	2	51	10	2	81	20	1
22	35	2	52	25	2	82	35	2
23	10	2	53	25	1	83	30	2
24	25	2	54	25	1	84	15	2
25	20	2	55	25	2	85	20	1
26	25	2	56	35	1	86	15	1
27	10	1	57	25	1	87	15	2
28	15	2	58	15	2	88	25	2
29	20	1	59	20	1	89	15	2
30	15	1	60	15	2	90	10	1

Kontainer ukuran 40								
No	Berat	Tujuan	No	Berat	Tujuan	No	Berat	Tujuan
1	20	2	15	20	2	29	20	2
2	10	1	16	35	2	30	10	2
3	15	2	17	25	1	31	15	2
4	30	2	18	25	2	32	20	2
5	25	2	19	30	2	33	35	2
6	15	1	20	15	2	34	30	2
7	20	2	21	10	2	35	35	2
8	15	2	22	30	2	36	30	2
9	10	1	23	25	1	37	25	2
10	20	2	24	10	2	38	35	1
11	10	2	25	10	1	39	35	1
12	10	1	26	30	2	40	10	1
13	25	2	27	10	1			
14	25	2	28	10	1			

B.4 Kasus 4

Kontainer ukuran 20								
No	Berat	Tujuan	No	Berat	Tujuan	No	Berat	Tujuan
1	10	2	33	25	2	65	20	1
2	15	1	34	15	1	66	35	1
3	15	1	35	10	2	67	15	2
4	35	2	36	15	2	68	10	1
5	15	2	37	25	1	69	30	1
6	20	2	38	25	2	70	10	2
7	35	1	39	20	1	71	25	1
8	10	1	40	10	2	72	20	1
9	25	2	41	30	2	73	15	2
10	25	1	42	15	1	74	10	2
11	10	1	43	35	1	75	20	1
12	10	1	44	25	2	76	30	2
13	30	1	45	15	1	77	25	2
14	15	2	46	15	2	78	10	1
15	20	2	47	15	1	79	15	2
16	10	2	48	15	2	80	15	1
17	15	1	49	25	1	81	20	1
18	25	2	50	30	2	82	20	1
19	15	1	51	25	2	83	10	2
20	15	2	52	15	1	84	35	2
21	30	2	53	35	1	85	25	2
22	15	1	54	25	1	86	10	1
23	20	2	55	15	2	87	10	2
24	35	2	56	15	2	88	25	1
25	20	2	57	25	2	89	15	2
26	10	1	58	10	1	90	20	1
27	10	2	59	25	1	91	35	2
28	15	2	60	35	2	92	20	1
29	10	2	61	20	2	93	10	1
30	20	1	62	35	2	94	35	2
31	15	2	63	15	2	95	30	1
32	25	2	64	15	1			

Kontainer ukuran 40								
No	Berat	Tujuan	No	Berat	Tujuan	No	Berat	Tujuan
1	25	2	16	35	1	31	10	2
2	20	1	17	20	1	32	10	1
3	20	2	18	20	2	33	35	1
4	15	2	19	25	1	34	20	1
5	20	2	20	10	1	35	25	2
6	20	2	21	10	1	36	20	2
7	20	1	22	35	1	37	25	2
8	20	2	23	15	2	38	15	1
9	30	1	24	35	2	39	10	2
10	30	2	25	35	1	40	10	2
11	10	2	26	15	1	41	20	2
12	35	1	27	25	2	42	35	2
13	15	1	28	30	2	43	10	1
14	25	1	29	25	2	44	35	1
15	25	2	30	15	2	45	30	1

B.5 Kasus 5

Kontainer ukuran 20								
No	Berat	Tujuan	No	Berat	Tujuan	No	Berat	Tujuan
1	35	2	33	15	1	65	15	1
2	15	2	34	15	3	66	25	1
3	35	1	35	10	1	67	10	3
4	20	3	36	10	3	68	15	2
5	15	1	37	35	1	69	10	3
6	15	1	38	20	1	70	15	1
7	15	3	39	20	1	71	25	3
8	20	2	40	30	3	72	15	3
9	20	2	41	25	2	73	35	3
10	35	1	42	10	2	74	25	3
11	20	1	43	10	3	75	25	1
12	30	1	44	30	2	76	25	1
13	30	3	45	15	1	77	10	3
14	15	1	46	10	2	78	20	3
15	25	1	47	15	2	79	10	1
16	25	1	48	35	3	80	20	2
17	20	2	49	15	2	81	25	2
18	35	3	50	20	2	82	25	1
19	35	3	51	10	3	83	15	3
20	10	3	52	35	3	84	15	1
21	15	2	53	35	3	85	10	3
22	15	3	54	20	2	86	25	1
23	35	3	55	30	2	87	15	2
24	20	2	56	30	3	88	20	1
25	15	1	57	25	3	89	10	2
26	35	3	58	25	2	90	35	2
27	20	2	59	20	3	91	25	2
28	15	2	60	25	1	92	10	2
29	30	1	61	15	3	93	25	3
30	15	3	62	35	3	94	30	3
31	10	1	63	20	1	95	35	3
32	10	1	64	25	3			

Kontainer ukuran 40								
No	Berat	Tujuan	No	Berat	Tujuan	No	Berat	Tujuan
1	25	1	16	15	2	31	10	3
2	15	1	17	10	2	32	10	1
3	30	1	18	25	1	33	10	1
4	15	2	19	15	1	34	15	1
5	20	1	20	20	1	35	10	1
6	20	2	21	20	3	36	25	3
7	25	1	22	35	2	37	35	2
8	15	3	23	10	1	38	20	3
9	10	1	24	35	2	39	15	1
10	35	2	25	30	1	40	20	2
11	15	2	26	15	3	41	15	3
12	20	3	27	30	1	42	20	3
13	25	2	28	25	3	43	25	1
14	10	3	29	15	3	44	25	2
15	10	1	30	10	2	45	25	3

B.6 *Unloading Time*

	Row1	Row2	Row3	Row4
Tier 1	144	150	156	162
Tier 2	138	144	150	156
Tier 3	132	138	144	150
Tier 4	126	132	138	144
Tier 5	120	126	132	138

BIOGRAFI PENULIS

Penulis lahir di Sumenep, tanggal 20 Juni 1990 dan diberi nama oleh kedua orang tuanya MATSAINI. Penulis merupakan anak pertama dari tiga bersaudara. Riwayat pendidikan Penulis. Sekolah Dasar di SDN Kolo-Kolo 1 Arjasa Kangean Sumenep (1997-2003). Kemudian melanjutkan ke SMPN 1 Arjasa Kangean Sumenep (2003-2006). Kemudian melanjutkan ke SMAN 1 Arjasa Kangean Sumenep (2006-2009). Setelah lulus Sekolah Menengah Pertama Penulis berkesempatan menimba ilmu di Perguruan Tinggi Negeri Pertama di Madura yaitu Universitas Trunojoyo Madura Jurusan Teknik Industri (2009-2013) dan memperoleh gelar Sarjana Teknik (S.T), setelah lulus Penulis membantu orang tua bekerja dan sambil kursus bahasa ingris demi mempersiapkan untuk meraih gelar yang lebih tinggi, akhirnya penulis berkesempatan menimba ilmu di Institut Teknologi Sepuluh Nopember Surabaya (2014-2017) dan memperoleh gelar Master Teknik (M.T). Email: matsay.abdullah@gmail.com