PENERAPAN HIBRIDISASI METODE *SIMULATED ANNEALING* DAN *IMPROVED GENETIC ALGORITHM* UNTUK OPTIMASI PENJADWALAN PRODUKSI-DISTRIBUSI

TESIS

Untuk memenuhi sebagian persyaratan
memperoleh gelar Magister Komputer

Disusun oleh:
Rafiuddin Rody
NIM: 176150100111007

PROGRAM STUDI MAGISTER ILMU KOMPUTER
JURUSAN TEKNIK INFORMATIKA
FAKULTAS ILMU KOMPUTER
UNIVERSITAS BRAWIJAYA
MALANG
2019

DAFTAR ISI

BAB 1 PENDAHULUAN

1.1 Latar Belakang

Semakin banyak perusahaan saat ini yang mengadopsi model bisnis *make to order* di mana produk dibuat sesuai pesanan (*order*) kemudian dikirim ke pelanggan dalam jeda waktu (*idle time*) yang sangat singkat (Chen, 2010). Pengiriman produk segera setelah selesainya proses produksi merupakan hal yang sangat penting terutama untuk produk yang sensitif terhadap waktu (*time sensitive product*). Sebagai contoh produksi dan pengiriman produk yang karakteristiknya mudah rusak seperti campuran beton siap pakai (Garcia & Lozano, 2004, 2005). Produk tersebut memiliki waktu guna yang sangat pendek. Setelah bahan baku campuran beton yang diperlukan dicampur, campuran yang dihasilkan mulai membeku dalam satu jam dan apabila campuran beton tidak digunakan maka akan menjadi tidak berguna. Produk seperti ini biasanya dikirimkan ke pelanggan segera setelah selesai diproduksi. Dalam hal ini, operasi produksi-distribusi sangat berkaitan erat dan hanya memiliki sedikit jeda waktu (*idle time*) di antara kedua proses tersebut atau bahkan tidak memiliki jeda waktu sama sekali. Contoh produk lainnya yang sensitif terhadap waktu adalah pencetakan-distribusi surat kabar (Van Buer, Woodruff, & Olson, 1999) dan percetakan-distribusi surat (Wang, Batta, & Szczerba, 2005). Dilihat dari karakteristik produk yang sensitif terhadap waktu, maka apabila penjadwalan produksi-distribusinya tidak diatur dengan baik akan mengakibatkan kerugian baik berupa uang maupun rusaknya reputasi perusahaan.

Beberapa model penjadwalan produksi-distribusi lain dalam literatur terdahulu yang didasarkan pada model bisnis *make-to-order* diantaranya adalah perakitan dan pengiriman produk elektronik seperti komputer (Li, Ganesan, & Sivakumar, 2006; Stecke & Zhao, 2007) dan penyiapan dan pengiriman makanan pada layanan katering makanan (Chen & Vairaktarakis, 2005). Dari beberapa contoh tersebut, dapat diketahui bahwa sering kali pesanan pelanggan merupakan barang pesanan khusus (*customized product*). Perusahaan sering kali harus memproses dan mengirimkan pesanan kepada pelanggan dalam waktu yang cepat. Hal ini membuat waktu pemrosesan pesanan dan waktu operasi pengiriman mempunyai keterkaitan sangat erat sehingga operasi produksi-distribusi ini harus dijadwalkan secara bersamaan (terintegrasi) untuk mengoptimalkan layanan dan biaya operasi.

Banyak peneliti telah mempelajari masalah produksi-distribusi terintegrasi dari berbagai aspek. Salah satu aspek terkait masalah ini yaitu masalah penjadwalan untuk pesanan (*order)* yang memiliki batas tanggal jatuh tempo (*due*

date). Keterlambatan pengiriman produk tidak hanya menimbulkan penalti keterlambatan karena ketidakpuasan pelanggan atau pelanggaran kontrak, tetapi juga menyebabkan kegagalan rantai pasok barang (*supply chain*). Tentu saja semua dampak tersebut juga akan mengurangi reputasi sebuah perusahaan. Di sisi lain, produk jadi yang dikirim ke pelanggan sebelum waktu jatuh tempo dapat mengakibatkan biaya penyimpanan tambahan (*additional inventory cost*), asuransi, atau bahkan mengakibatkan kerusakan produk. Oleh karena itu, masalah penjadwalan terintegrasi yang melibatkan pertimbangan tanggal jatuh tempo menjadi sangat penting untuk sebagian besar bisnis yang ada saat ini.

Wang, Grunder dan Moudni (2014) dalam penelitiannya mencontohkan permasalahan produksi-distribusi terintegrasi dengan model satu pabrik dan satu atau lebih pelanggan, lebih lanjut dijelaskan bahwa setiap pesanan yang diminta oleh pelanggan memiliki tanggal jatuh tempo yang berbeda. Tujuannya adalah mengoptimalkan fungsi objektif yang meminimalkan total biaya produksi-distribusi. Selain itu, mereka juga menunjukkan bahwa masalah yang ditelitinya adalah *NP-Hard*. Sebuah persoalan dikatakan *NP-Hard* jika tidak mungkin diselesaikan menggunakan *polynomial-time algorithm*. Lebih lanjut, Wang, Grunder dan Moudni (2014) mengusulkan salah satu metode metaheuristik yaitu algoritme genetika untuk memecahkan masalah tersebut.

Metode heuristik dan metaheuristik dalam ilmu komputer merupakan metode pencarian solusi untuk mencari pendekatan dari sebuah solusi dalam masalah optimasi. Tujuannya adalah untuk menemukan solusi optimal dari semua kemungkinan solusi, yaitu salah satu solusi yang meminimalkan atau memaksimalkan fungsi objektif. Fungsi objektif adalah fungsi yang digunakan untuk mengevaluasi kualitas solusi yang dihasilkan. Kumpulan semua solusi yang memungkinkan untuk masalah yang diberikan dapat dianggap sebagai ruang pencarian, dan algoritme pengoptimalan, sering disebut sebagai algoritme pencarian (Kokash, 2014).

Metode heuristik dan metaheuristik banyak digunakan dalam penyelesaian permasalahan penjadwalan dengan karakteristik *NP-Hard*. Beberapa penelitian terkait produksi-distribusi yang menggunakan metode heuristik dapat dilihat pada (Anshulika & Bewoor, 2017; Ruiz & Pan, 2016; Supithak, Liman, & Montes, 2010). Dalam penerapannya, metode heuristik sering kali terjebak dalam lokal optimal (*local optimum*) ketika masalah yang dikerjakan berukuran besar, sehingga pendekatan metaheuristik secara luas digunakan (Mhasawade & Bewoor, 2017). Dalam permasalahan optimasi dibidang ilmu komputer, sebuah lokal optimal diartikan sebagai sebuah solusi yang optimal (maksimal atau minimal) dalam sekumpulan kandidat solusi yang berdekatan (tetangga). Ini berbeda dengan global optimal (*global optimum*), yang merupakan

solusi optimal di antara semua solusi yang mungkin, tidak hanya di lingkungan solusi tetangga.

Dalam perkembangan selanjutnya, pendekatan hibridisasi digunakan untuk menigkatkan kualitas pencarian solusi. Hibridisasi metaheuristik digunakan untuk mencari solusi optimal dalam waktu yang wajar dan meningkatkan performa dengan cara mengkombinasikan keunggulan-keunggulan metode metaheuristik yang diharapankan dapat menghindari lokal optimal dan memperoleh global optimal. Salah satu penerapan hibridisasi algoritme metaheuristik untuk menyelesaikan permasalah penjadwalan produksi-distribusi menggunakan algoritme genetika dan *simulated annealing* telah dilakukan oleh Wang dan Luo (2016).

Pada penelitian ini penulis berfokus pada permasalahan penentuan ukuran pesanan bahan baku (*lot-sizing*) dan penjadwalan produksi-distribusi dengan model bisnis *make to order* dengan jumlah (*volume*) pesanan dan tanggal jatuh tempo (*due date*) yang berbeda-beda. Dalam permasalahan ini, pesanan-pesanan diproduksi pada mesin produksi secara berkelompok (*batch*) dan kemudian dikirim ke pelanggan menggunakan kendaraan yang memiliki kapasitas tertentu (*capacity*). Setiap pesanan yang memiliki tanggal jatuh tempo dan jumlah yang berbeda tersebut harus dikirimkan ke pelanggan sebelum tanggal jatuh tempo (keterlambatan tidak diperbolehkan). Waktu yang dibutuhkan dalam satu kali pemrosesan pesanan dalam *batch* merupakan suatu nilai yang tetap dan tidak bergantung pada jumlah total pesanan (*volume*) di dalamnya. Dalam proses produksi, penentuan waktu *set-up* serta biaya *set-up* diperlukan sebelum *batch* diproses. Dalam pengiriman, penentuan waktu pengiriman (*delivery time*) serta biaya pengiriman (*delivery cost*) diperlukan untuk setiap pengiriman (pulang-pergi) antara pabrik dan pelanggan. Selain itu, diasumsikan bahwa pesanan yang tiba di pelanggan sebelum tanggal jatuh tempo akan menimbulkan biaya persediaan pelanggan (*inventory cost*). Tujuannya adalah untuk menemukan skema ukuran produksi pesanan dan penjadwalan produksi-distribusi dimana biaya total (*total cost*) dapat diminimalkan sekaligus menjamin kualitas layanan kepada pelanggan. Perusahaan yang menerapkan model produksi menggunakan mesin *batching* untuk produksi barang dengan model *make to order* masih sangat jarang, akan tetapi kedepannya model produksi *make to order* yang dikerjakan secara tumpukan (*batch*) akan semakin menjadi *trend*. Salah satu contoh perusahaan di kota Malang yang menerapkan model produksi *make to order* menggunakan mesin *batching* adalah PT Agaricus Sido Makmur Sentosa (ASIMAS).

Wang, Grunder dan Moudni (2014) menggunakan algoritme genetika yang telah dimodifikasi untuk menyelesaikan permasalahan *lot-sizing* dan penjadwalan-distribusi dengan model permasalahan seperti yang telah dijelaskan sebelumnya dan menghasilkan rata-rata *error* tidak lebih dari 12%. Pada

penelitiannya yang lain, Wang dan Luo (2016) mencoba menyelesaikan permasalahan serupa menggunakan hibridisasi metode *simulated annealing* dan algoritme genetika yang standar akan tetapi menghasilkan rata-rata *error* lebih tinggi yaitu 16%. Penerapan hibridisasi metode diharapkan dapat meningkatkan performa sebuah metode metaherusitik, akan tetapi dalam kasus ini hal tersebut tidak terjadi. Sehingga, pada penelitian ini penulis mengusulkan penggunaan hibridisasi metode *simulated annealing* dan *improved genetic algorithm* untuk menyelesaikan permasalahan *lot-sizing* dan penjadwalan produksi-distribusi dan diharapkan dapat diperoleh hasil yang lebih baik dari kedua metode tersebut. Permasalahan penjadwalan produksi-distribusi ini akan dimodelkan ke dalam *mixed integer programing*, dan kemudian metode hibridisasi *simulated annealing* dan *improved genetic algorithm* dikembangkan untuk menyelesaikan model tersebut. Selanjutnya metode *lower bound* digunakan untuk evaluasi kinerja hibridisasi metode pada hasil yang diperoleh.

1.2 Rumusan Masalah

Bedasarkan latar belakang penelitian, maka diperoleh rumusan masalah dalam penelitian ini adalah sebagai berikut.

1. Bagaimana representasi permasalahan produksi-distribusi yang menerapkan model sistem bisnis *make-to-order* dalam bentuk model *mixed integer programming* (MIP)?
2. Bagaimana merancang dan menerapkan hibridisasi metode *simulated annealing* dan *improved genetic algorithm* untuk menyelesaikan permasalahan Model MIP?
3. Bagaimana tingkat akurasi penjadwalan yang dihasilkan menggunakan hibridisasi metode *simulated annealing* dan *improved genetic algorithm* setelah dievaluasi menggunakan metode *lower bound*?
4. Berapa perbedaan akurasi hasil penerapan hibridisasi metode *simulated annealing* dan *improved genetic algorithm* untuk menyelesaikan permasalahan Model MIP dibandingkan dengan metode penjadwalan lainnya?

1.3 Tujuan

Berdasarkan rumusan masalah yang telah dibuat, maka tujuan dari penelitian ini adalah sebagai berikut:

1. Mengetahui representasi permasalahan produksi-distribusi yang menerapkan model sistem bisnis *make-to-order* dalam bentuk model *mixed integer programming* (MIP).

2. Memperoleh bentuk rancangan dan penerapan hibridisasi metode *simulated annealing* dan *improved genetic algorithm* untuk menyelesaikan permasalahan model MIP.
3. Mengetahui tingkat akurasi penjadwalan yang dihasilkan menggunakan hibridisasi metode *simulated annealing* dan *improved genetic algorithm* setelah dievaluasi menggunakan metode *lower bound*.
4. Memperoleh perbandingan tingkat akurasi penerapan hibridisasi metode *simulated annealing* dan *improved genetic algorithm* untuk menyelesaikan permasalahan model MIP dibandingkan dengan metode penjadwalan lainnya.

1.4 Manfaat

Manfaat yang diharapkan diperoleh dari hasil penulisan tesis ini antara lain:

1. Perusahaan atau badan usaha yang menerapkan model sistem bisnis *make to order* diharapkan dapat memperoleh penjadwalan produksi-distribusi yang menghasilkan biaya optimal apabila menerapkan metode yang diusulkan ini dan diharapkan juga dapat memberikan kepuasan pelanggan.
2. Dengan mengetahui perbandingan tingkat akurasi penjadwalan yang dihasilkan, diharapkan mampu memberikan kontribusi terhadap metode yang dapat digunakan dalam penyelesaian permasalahan penjadwalan produksi-distribusi yang menerapkan model sistem bisnis *make to order*.

1.5 Batasan Masalah

Batasan masalah yang terdapat pada penelitian ini, diantaranya:

1. Permasalahan penjadwalan yang diteliti merupakan penjadwalan yang menerapkan model sistem bisnis *make to order*.
2. Model penjadwalan merupakan penjadwalan dengan satu pabrik dan satu atau banyak pelanggan.
3. Data yang dipakai dalam penelitian merupakan data acak dan *artificial* yang dibangkitkan dengan mengacu pada literatur penelitian yang dilakukan oleh Wang dan Lou (2016).

1.6 Sistematika Pembahasan

Sistematika pembahasan berisi mengenai susunan penelitian dimulai dari pendahuluan sampai penutup yang telah disesuaikan dengan tata cara penulisan laporan yang berlaku di Fakultas Ilmu Komputer Universitas Brawijaya Malang. Adapun tujuan ditulisnya sistematika pembahasan ini supaya pembaca dapat memahami sistematika dalam penelitian ini.

BAB 1 PENDAHULUAN

Pendahuluan merupakan gambaran secara umum menganai latar belakang permasalahan, rumusan masalah, tujuan, manfaat, batasan masalah dan sistematika penulisan.

BAB 2 LANDASAN TEORI

Pada bagian landasan teori akan diuraikan dasar teori dan kajian pustaka yang berkaitan dengan permasalahan yang dikaji, metode-metode yang telah digunakan pada penelitian sebelumnya untuk permasalahan penjadwalan produksi. Dan kemudian dasar metode *simulated annealing* dan *algoritme genetika* sebagai metode yang diusulkan dalam proses penjadwalan produksi-distribusi yang menerapkan model sistem bisnis *make to order*.

BAB 3 METODOLOGI

Metodologi berisi tentang alur penelitian untuk menyelesaikan permasalahan penjadwalan produksi-distribusi yang menerapkan model sistem bisnis *make-to-order* yang dimulai dari metode penelitian hingga skenario pembanding.

BAB 4 PERANCANGAN DAN IMPLEMENTASI

Bagian implementasi meliputi penggabungan antara perancangan antarmuka, perancangan model MIP, perancangan hibridisasi metode, dan perancangan uji metode untuk menyelesaikan model MIP dalam bentuk representasi kromosom.

BAB 5 PENGUJIAN DAN PEMBAHASAN

Pengujian dan pembahasan membahas mengenai tata cara pengujian solusi dari penerapan hibridisasi metode untuk menyelesaikan model MIP serta menganalisa solusi yang dihasilkan.

BAB 6 PENUTUP

Penutup berisi kesimpulan dan saran dari hasil penerapan metode hibridisasi untuk menyelesaikan permasalahan penjadwalan produksi-distribusi.

BAB 2 LANDASAN TEORI

2.1 Penjadwalan Produksi

2.1.1 Make to Order

Menurut Stecke dan Zhao (2007) terdapat beberapa perusahaan yang memproduksi produk dan menyimpannya sebagai persediaan sampai produk tersebut dijual. Model produksi seperti ini biasa disebut *make to stock*. Sebaliknya ada juga beberapa perusahaan lainnya yang hanya memproduksi produk setelah ada pesanan dari pelanggan. Model produksi yang terakhir ini biasa disebut *make to order*. Model produksi *make to order* mempunyai kelebihan dan kekurangan. Keuntungan dari model produksi ini yaitu perusahaan dapat mengurangi biaya penyimpanan barang (*inventory*) dan memungkinkan untuk memproduksi produk dengan menyesuaikan pesanan pelanggan. Namun, dari sisi lain, pelanggan harus menunggu pesanan selesai diproduksi dan dikirim. Hal lain yang perlu diperhatikan adalah adanya kemungkinan pesanan sampai ke pelanggan melampaui tanggal jatuh tempo karena ketidaksesuaian penjadwalan produksi atau kapasitas produksi yang tidak mencukupi. Permasalahan di atas merupakan salah satu alasan mengapa penelitian ini dilakukan.

2.1.2 Permasalahan Penjadwalan Produksi

Pada bagian ini akan dipaparkan beberapa penelitian terkait permasalahan dalam penjadwalan produksi.

2.1.2.1 Flow Shop Problem

Permasalahan penjadwalan *Flow Shop* telah dipelajari secara luas lebih dari 10 tahun terakhir (Neufeld, Gupta, & Buscher, 2016). Sebuah sistem *flow shop* terdiri dari mesin *m* yang disusun secara seri di mana satu set pekerjaan (*job*) *n* harus melalui mesin pertama, kemudian mesin dua, dan seterusnya sampai mesin *m*. Artinya, semua pekerjaan memiliki jalur yang sama. Ketika sedang diperoses, setiap pekerjaan membutuhkan waktu pemrosesan tanpa disela pada masing-masing mesin. Setiap pekerjaan hanya dapat dikerjakan di satu mesin pada saat yang sama. Proses pekerjaan pada mesin tidak dapat disela. Semua pekerjaan tidak terikat satu dengan yang lain (*independent*) dan tersedia untuk diproses pada waktu ke-0. Tujuannya adalah untuk menemukan urutan pemrosesan pekerjaan pada semua mesin sehingga dapat meminimalkan *make span* dan *flow time*. Penelitian berikut membahas permasalahan *flow shop* yang diselesaikan menggunakan metode heuristik diantaranya: *Harmony search algorithm*

(Akkoyunlu, Engin, & Buyukozkan, 2015), *Hibridisasi* GA-HS (Yun Bao, Hua Jing, & Liping Zheng, 2011), *Genetic Algorithm* (Xudong Wang & Qingyun Dai, 2014) dan *Simulated Annealing* (Ji, He, & Wang, 2009).

2.1.2.2 *Job* Shop Problem

Secara umum masalah penjadwalan *Job Shop* dapat digambarkan sebagai berikut: satu set pekerjaan (*job*) *n* harus diproses pada satu set mesin *m* yang sudah tersedia dari waktu ke-0, dan setiap pekerjaan membutuhkan teknik pemrosesan khusus. Setiap pekerjaan terdiri dari sebuah urutan operasi, dan masing-masing operasi akan dikerjakan dengan menggunakan salah satu mesin tanpa disela. Setiap operasi harus segera diproses setelah selesai diproses pada operasi sebelumnya. Tujuan dari masalah ini adalah untuk menemukan jadwal produksi yang meminimalkan *makespan* (Cmax) atau mengoptimalkan target lainnya, dengan cara menentukan waktu mulai produksi (*starting time*) dan urutan mesin untuk setiap pekerjaan (Mhasawade & Bewoor, 2017).

Penelitian lainnya berkaitan dengan permasalahan *Job Shop* dilakukan oleh De Giovanni dan Pezzella (De Giovanni & Pezzella, 2010) menggunakan algoritme genetika, beberapa penelitian lain diselesaikan menggunakan *Bee Colony Algorithm* (Lvshan, Dongzhi, & Weiyu, 2017; Ming Huang, Yao Bai, & Xu Liang, 2012) dan *tabu search* (Kawaguchi & Fukuyama, 2016).

2.1.2.3 Lot-sizing Problem

Lot-sizing didefinisikan sebagai jumlah barang yang harus diproduksi atau dipesan. Sebuah perusahaan manufaktur yang ingin berkompetisi di pasar harus membuat keputusan yang tepat dalam permasalahan *lot-sizing* yang berdampak langsung pada kinerja dan produktivitas sistem (Aouadni & Rebai, 2013). Hal ini merupakan keputusan penting yang harus diambil oleh produsen mana pun.

Tujuan dari masalah *lot-sizing* adalah untuk menentukan jumlah pesanan bahan baku dalam satu periode untuk memenuhi permintaan pelanggan dengan meminimalkan total pemesanan, pembelian, dan biaya penyimpanan. Ukuran *lot-size* yang lebih kecil akan mengurangi biaya penyimpanan, tetapi meningkatkan biaya pemesanan. Di lain sisi, ukuran *lot-size* yang lebih besar akan mengurangi biaya pemesanan akan tetapi akan menaikan biaya penyimpanan. Beberapa peneliti mencoba untuk menyelesaikan permasalahan *lot-sizing* menggunakan metode heuristik dan memperoleh hasil yang cukup baik (Supithak et al., 2010; D. Wang, Guo, & Zhu, 2014).

2.1.2.4 Batasan Waktu (*due date*)

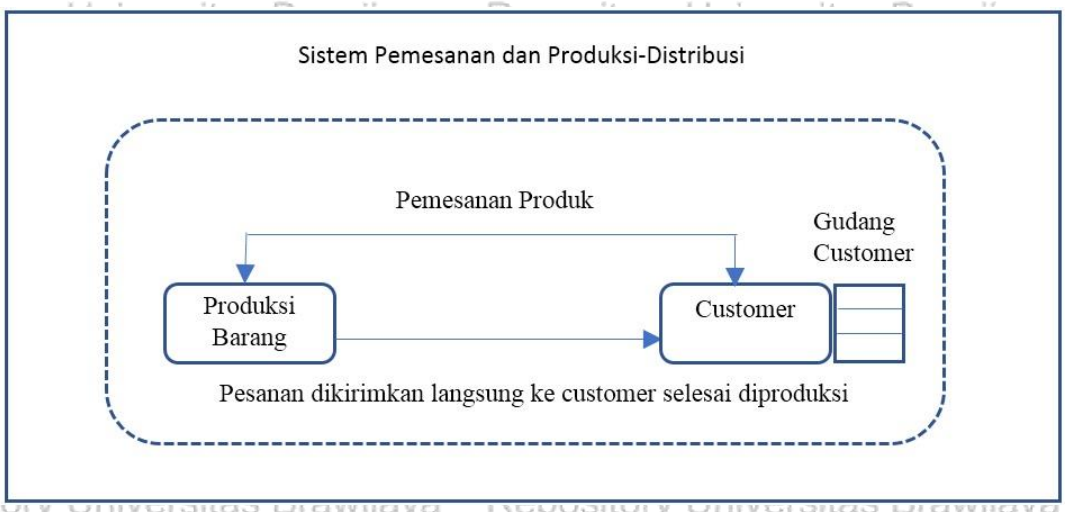
Dengan meningkatnya minat produsen industri untuk menerapkan konsep *Just-In-Time*, banyak penelitian yang bertujuan untuk meminimalkan keterlambatan pengiriman barang pesanan (*tardiness*) atau terlalu cepatnya barang dikirim (*earliness*) dari batas waktunya (Shabtay, 2016). Masalah seperti ini muncul pada penjadwalan di mana pelanggan tidak menginginkan menerima pesanan mereka baik lebih awal atau lebih lambat dari batas waktu (*due date*). Ini dikarenakan, apabila suatu pesanan tiba sebelum tanggal jatuh temponya, maka pelanggan harus menyimpan pesanan di gudang mereka (*inventory*) sampai tanggal batas waktunya. Akibatnya, barang pesanan ini menimbulkan biaya penyimpanan yang besarnya bergantung pada berapa lama barang pesanan disimpan. Denda biaya *earliness* ini dapat timbul karena menurunnya kualitas barang, biaya penyimpanan, asuransi dan variable lainnya. Begitu juga sebaliknya, apabila pekerjaan tiba setelah tanggal jatuh tempo, itu menimbulkan biaya penalti keterlambatan. Penalti keterlambatan dapat diakibatkan oleh ketidakpuasan pelanggan, denda dalam kontrak, dan adanya potensi kehilangan reputasi (Chen, 1996).

2.1.2.5 *Batch* Processing Machine

Batch Processing Machine (BPM) adalah mesin produksi yang dapat secara bersamaan memproses beberapa pekerjaan dalam satu tumpukan (*batch*). Mesin ini umumnya digunakan pada oven pengolah panas, operasi pembakaran semikonduktor, pemprosesan kimia di dalam tangki atau kiln, pengujian proses sirkuit listrik, atau industri bahan farmasi dan konstruksi (Esmaeili, Molla-Alizadeh-Zavardehi, Mahmoodirad, & Niroomand, 2015).

2.2 Deskripsi Permasalahan

Makalah ini mengkaji permasalahan penjadwalan produksi-distribusi dengan model rantai pasok (*supply chain*) *make-to-order* dimana produksi dan distribusi diproses secara langsung tanpa melibatkan gudang penyimpanan (*inventory*. Penjadwalan produksi-distribusi ini memiliki model permasalahan dengan jumlah (*volume*) pesanan dan tanggal jatuh tempo (*due date*) yang berbeda-beda. Dalam permasalahan ini, pesanan-pesanan diproduksi pada mesin produksi secara berkelompok (*batch*) dan kemudian dikirim ke pelanggan menggunakan kendaraan yang memiliki kapasitas tertentu (*capacity*). Setiap pesanan yang memiliki tanggal jatuh tempo dan jumlah yang berbeda tersebut harus dikirimkan ke pelanggan sebelum tanggal jatuh tempo (keterlambatan tidak diperbolehkan). Berikut ini merupakan ilustrasi permasalahan yang dapat dilihat pada **Gambar 2.1**.

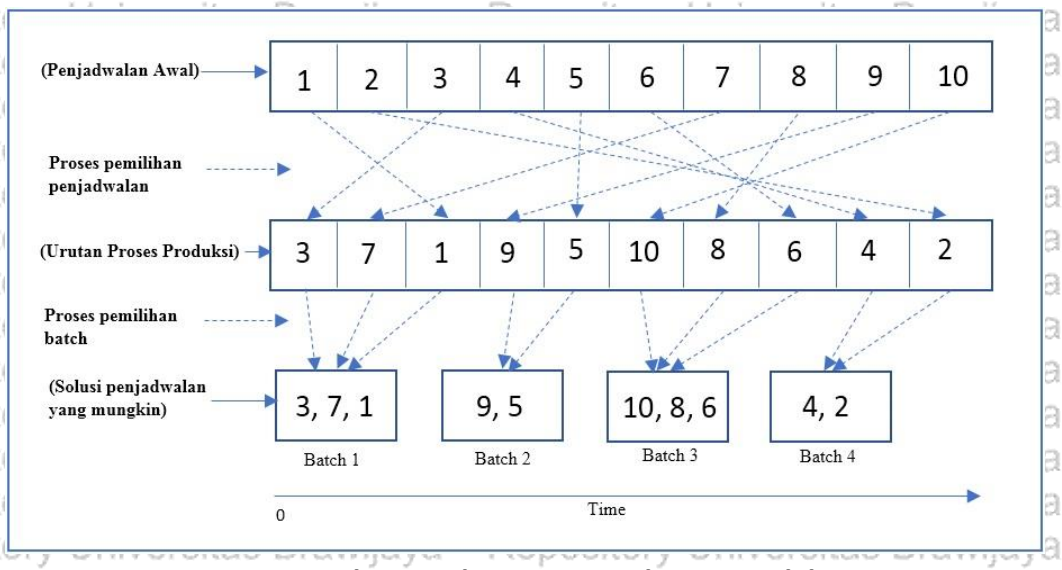


Gambar 2.1 Ilustrasi Masalah Yang Diusulkan

Pertama-tama supplier memperoleh pesanan (*order*) dari pelanggan. Semua pesanan yang ada kemudian dijadikan satu set pekerjaan ($J = \{j_1, j_2, \ldots, j_n\}$). Set pekerjaan pertama kali diproses pada mesin *batching*. Setiap pekerjaan memiliki tanggal jatuh tempo yang berbeda d_i (*due date*) dan jumlah yang berbeda v_i (*volume*). Sebuah mesin *batching* dapat memproses beberapa pekerjaan secara bersamaan dalam satu *batch* (tumpukan pekerjaan), dengan syarat total *volume* pekerjaan kurang dari kapasitas mesin *c*. Setiap *batch* yang akan diproses pada mesin *batching* menghasilkan λ_c (biaya setup) dan λ_t (waktu pengaturan). Waktu pengolahan satu *batch* pada mesin *batching* adalah konstan p_t yang tidak dipengaruhi oleh total *volume* yang didalamnya.

Setelah selesai diproduksi, pekerjaan langsung dikirimkan menggunakan kendaraan pengangkut (*transporter*) yang berkapasitas *c* (sama dengan kapasitas mesin *batching*) ke pelanggan. Diasumsikan bahwa setiap pekerjaan harus sampai kepada pelanggan sebelum atau pada tanggal jatuh tempo, dan keterlambatan pengiriman tidak diperbolehkan. Lebih lanjut, setiap pekerjaan yang dikirimkan ke pelanggan sebelum tanggal jatuh tempo akan menimbulkan biaya persediaan pelanggan (*inventory cost*). Setiap *batch* yang akan dikirim menggunakan kendaraan pengangkut memiliki η_c (biaya pengiriman) dan η_t (waktu pengiriman pulang-pergi).

Dalam permasalahan ini, produsen harus membuat dua keputusan. Pertama, menentukan urutan pemrosesan pekerjaan dalam satu set pekerjaan *J*. Kedua, menentukan pada *batch* mana pekerjaan tersebut diproses. Sebuah ilustrasi permasalahan yang diadopsi dari (Wang, Grunder, & Moudni, 2014), digambarkan pada **Gambar 2.2**.



Gambar 2.2 Ilustrasi Contoh Permasalahan

Angka-angka pada **Gambar 2.2** menunjukkan pesanan konsumen yang harus dijadwalkan. Pengambilan kedua keputusan di atas akan mempengaruhi biaya produksi-distribusi karena jumlah *batch* dipengaruhi oleh pengaturan pekerjaan (*job*) ke dalam *batch*. Disamping itu juga terdapat batasan waktu yang harus dipenuhi. Tujuannya adalah membuat sebuah model penjadwalan yang terkoordinasi sedemikian rupa sehingga biaya total produksi-distribusi dapat diminimalkan.

2.3 Penelitian Terkait

Pada bagian ini akan disebutkan beberapa penelitian terkait dengan penjadwalan produksi-distribusi dan penjelasan mengenai beberapa contoh metode heuristik yang digunakan.

2.3.1 Metode *Artificial Bee Colony*

Artificial Bee Colony (ABC) Algorithm merupakan salah satu algoritme heuristik yang didasarkan pada perilaku lebah ketika mencari sumber makanan di alam. Dalam algoritme ABC, koloni buatan dibagi menjadi tiga jenis yang berbeda: lebah pekerja (*employed bee*), lebah pengawas (*onlooker bee*) dan lebah pengintai (*scout bee*) dan dalam algoritme ini kemudian diperkenalkan tiga pola dasar perilaku lebah: mencari sumber makanan, merekrut lebah pengikut dan memberikan sumber makanan (Ming Huang et al., 2012). ABC dapat dikategorikan dalam algoritme heuristik berbasis populasi.

Ming Huang et al (Ming Huang et al., 2012) mengusulkan hibridisasi ABC dan Algoritme genetika dengan cara menambahkan operasi mutasi dan cross over ke dalam operasi ABC. Melalui operasi mutasi dan *crossover*, ABC dapat

menghasilkan solusi yang luas dan lebih bervariasi. Penggabungan kedua algoritme ini dapat meningkatkan efisiensi penyelesaian masalah penjadwalan.

Hibridisasi ABC dan Algoritme Genetik juga dilakukan oleh Lvshan et al (Lvshan et al., 2017). Dalam proses penggabungan ini, hanya operasi mutasi yang diaplikasikan ke dalam ABC. Lvshan et al menggunakan operasi mutasi untuk menutupi kekurangan ABC dalam pencarian solusi lokal.

2.3.2 Metode *Particle Swarm Optimization*

Particle swarm optimization (PSO) merupakan metode optimasi global heuristik yang didasarkan kecerdasan selekompok hewan. Algoritme ini dihasilkan dari penelitian perilaku sekawanan burung dan kawanan ikan. Saat mencari makanan, burung-burung bisa menyebar atau pergi berkelompok sebelum mereka menemukan tempat di mana ada sumber makanan. Ketika burung-burung mencari makanan dari satu tempat ke tempat lain, selalu ada satu atau beberapa burung yang dapat mencium bau makanan dengan sangat baik. Hal ini menandakan bahwa burung tersebut memiliki informasi sumber makanan yang lebih baik dan di sekitar burung tersebut dapat dianggap sebagai tempat di mana makanan dapat ditemukan. Karena mereka menyebarkan informasi ketika mencari makanan dari satu tempat ke tempat lain, burung-burung pada akhirnya akan berduyun-duyun ke tempat di mana makanan dapat ditemukan. (Choubey, 2017). Karena penerapannya yang mudah, algoritme PSO banyak digunakan dalam penelitian dan berkembang dengan cepat. PSO merupakan algoritme heuristik berbasis populasi.

Dalam penelitian (Wu, Zhou, & Li, 2009), algoritme PSO digunakan untuk menentukan penjadwalan produksi dari permintaan pelanggan (*demand*) di setiap pabrik manufaktur.

Ping dan Minghai (2014) mengusulkan PSO yang dimodifikasi untuk menyelesaikan permasalahan penjadwalan produksi. Ping dan Minghai menambahan element *chaotic local search strategy* untuk meningkatkan performa pencarian pada solusi yang menjanjikan.

PSO dan algoritme genetika digunakan Mudjihartono et al (Mudjihartono, Jiamthapthaksin, & Tanprasert, 2016) secara pararel untuk menyelesaikan permasalahan penjadwalan produksi. Ide yang diterapakan adalah pergerakan partikel didasarkan pada dua nilai: perbedaan jarak dan nilai *fitness*. Perbedaan jarak menggambarkan bagaimana partikel berbeda dari solusi global terbaik sementara perbedaan nilai *fitness* menggambarkan seberapa jauh kebugaran partikel solusi global terbaik. Berdasarkan nilai-nilai ini, setiap partikel memiliki probabilitas sendiri untuk mutasi atau *crossover*. Semakin besar perbedaan jaraknya, semakin besar kemungkinan partikel akan dipindahkan.

2.3.3 Metode Tabu Search

Tabu Search (TS) menerapkan metode pencarian lokal (*local search*) dan meningkatkan solusi dengan cara menghasilkan solusi tetangga (*neighborhood solution*) dari solusi sekarang (*current solution*) dan memindahkan solusi tetanggal terbaik (*best neighborhood solution)*, yang tidak tabu (terlarang). Metode pencarian lokal harus dihentikan apabila semua solusi tetangga tidak lebih baik daripada solusi sekarang. TS dapat melanjutkan pencarian dengan memindahkan solusi tetangga terbaik yang tidak tabu, meskipun jika solusi tetangga terbaik tersebut tidak lebih baik dari solusi saat ini (Kawaguchi & Fukuyama, 2016). Karakteristik yang dimiliki TS ini memungkinkan pencarian solusi untuk keluar dari solusi optimal lokal (*local optimal*) dan menjelajahi ruang solusi yang belum pernah dicari sebelumnya .

Wang, Guo dan Zhu (2014) dalam penelitiannya memakai algoritme TS untuk menyelesaikan permasalahan penjadwalan produksi-distribusi terintegrasi dan menghasilkan hasil yang cukup baik.

Ada juga penelitian lainnya yang menggunakan hibridisasi algoritme tabu search dan Algoritme Genetika untuk menyelesaikan permasalahan penjadwalan produksi. Penelitian ini dilakukan oleh Liang Di dan Tao Ze (Liang Di & Tao Ze, 2011). Hibridisasi kedua algoritme ini secara efektif dapat mencegah munculnya konvergensi dini.

2.3.4 Metode *Genetic Algorithm*

Genetic algorithm (GA) adalah algoritme optimisasi stokastik yang awalnya dikembangkan oleh John Holland pada tahun 1975. Algoritme ini tidak berusaha untuk menemukan solusi analitik yang tepat tetapi lebih bertujuan untuk menemukan solusi yang memuaskan (mendekati optimal) dalam waktu perhitungan yang wajar (Aouadni & Rebai, 2013). Penjelasan lebih lengkap mengenai GA akan dijabarkan pada subbab 2.5 di bawah.

GA digunakan oleh Mahmudy et el untuk menyelesaikan permasalahan *Flexible Manufacturing System* menggunakan *Real Coded Genetic Algorithm (RCGA).* Hasil penelitian tersebut menunjukkan bahwa penerapan RCGA meskipun hanya dengan operasi sederhana, mampu menghasilkan hasil yang menjanjikan yang sebanding dengan yang dicapai oleh pendekatan-pendekatan dalam literatur lainnya (Mahmudy, Marian, & Luong, 2013a, 2013b).

Contoh lainnya dilakukan oleh Huang et el (Huang, Wang, & Liang, 2017), mereka mengusulkan GA yang telah dimodifikasi dan ditingkatkan untuk menyelesaikan permasalah penjadwalan produksi. Dalam penelitiannya diterapkan strategi pemilihan *roulette wheel* (acak dengan *probabilitas*) ketika membangkitkan populasi awal untuk meningkatkan kualitas populasi awal. Dalam

proses *crossover* dan mutasi, probabilitas kedua operasi ini dapat berubah secara adaptif. Hal ini dilakukan untuk menghindari konvergensi dini.

Banyak eksperimen dan laporan menunjukkan bahwa GA merupakan metode yang efisien dan kuat dalam penyelesaian masalah produksi. Namun, GA juga memiliki kekurangan. Jika kromosom terburuk dihilangkan setelah setiap generasi, maka populasi akan cenderung cepat menjadi homogen (konvergensi dini). Pendekatan hibridisasi GA dan *simulated annealing* (SA) digunakan Wang dan Lou (2016) untuk menyelesaikan permasalahan konvergensi dini tersebut. SA sesekali memungkinkan calon solusi bergerak ke arah tujuan yang kurang baik untuk menghindari lokal optima. Namun demikian, algoritme hibridisasi GA dan SA ini biasanya tidak lebih cepat dari pada algoritme genetik yang biasa.

Melihat keefektifan dan keunggulan algoritme genetika untuk menyelesaikan permasalahan penjadwalan produksi, baik menggunakan algoritme genetika murni, algoritme genetika yang dimodifikasi, maupun hibridisasi algoritme genetika dengan metode lainnya, maka penulis juga akan menggunakan algoritme genetika untuk menyelesaian permasalahan penjadwalan produksi-distribusi dengan model bisnis *make to order*.

2.4 *Mixed Integer Programming*

Integer Programming (IP) merupakan sebuah *linear programming* (LP) yang memenuhi syarat semua variabelnya bernilai bilangan bulat (*integer*). IP dapat dibagi menjadi tiga kategori tergantung pada nilai variabel keputusannya. Ketika semua variabel keputusan adalah bilangan bulat, maka disebut *Pure Integer Programming*. Jika semua variabel bernilai 0 atau 1 (*biner*), dikatakan sebagai *binary integer programming*. Apabila variable keputusan dari sebuah Pemrograman Integer model terdiri dari bilangan *integer* dan bilangan *biner,* maka model tersebut disebut *Mixed Integer Programming*. (Wang, Peng, & Hung, 2016)

2.5 Algoritme Genetika

Algoritme genetika (GA) banyak digunakan sebagai teknik pencarian berdasarkan teori *survival of fittest* oleh Darwin. Algoritme ini berbeda dari teknik pengoptimalan lainnya karena algoritme ini tidak bekerja secara langsung pada parameter permasalahan tetapi dalam bentuk yang telah dikodekan (Bhatt & Chauhan, 2015). Berikut adalah langkah-langkah dalam GA.

2.5.1 Representasi Kromosom

Langkah pertama untuk membangun atau menjalankan algoritme genetika adalah menyediakan pengkodean yang sesuai untuk solusi yang akan diselesaikan *(possible solution).* Pengkodean dari solusi yang dicari disebut sebagai kromosom. Pada setiap kromosom tersebut, terdapat fungsi kebugaran (nilai *fitness*) yang

nilainya harus dievaluasi (Bhatt & Chauhan, 2015). Nilai *fitness* sendiri merupakan sebuah acuan penilaian seberapa baik kromosom yang dihasilkan untuk dipilih menjadi calon solusi (Mahmudy, Marian, & Luong, 2012). Dalam algoritme genetika setiap kromosom disebut sebagai individu. Kumpulan Individu-individu mewakili sebuah populasi dan selama fase reproduksi orangtua dipilih dari populasi ini sehingga mereka dapat berproduksi untuk menghasilkan generasi berikutnya. Proses reproduksi dilakukan dengan dua cara yaitu menggunakan operator *crossover* (tukar silang) dan *mutation* (mutasi). Berikut terdapat beberapa pengkodean umum yang biasa digunakan untuk merepresentasikan solusi suatu permasalahan (Mahmudy, 2015):

a) Pengkodean Biner untuk Representasi Bilangan Bulat

Representasi kromosom dengan pengkodean biner dapat digunakan untuk merepresentasikan bilangan bulat dengan syarat banyaknya kemungkinan bilangan bulat tersebut sebesar pangkat dua bilangan tertentu (n^2), sehingga masing-masing kode biner dapat merepresentasikan semua bilangan bulat.

b) Pengkodean Biner untuk Representasi Bilangan Riil

Representasi dengan pengkodean biner dapat juga digunakan untuk merepresentasikan bilangan riil. Jika bilangan riil R berada dalam interval R_{min} dan R_{max}, maka pengkodean biner yang mungkin dapat dilakukan adalah dengan memetakan biner terendah (0 0 ... 0) dan tertinggi (1 1 ... 1) sebagai R_{min} dan R_{max}. Setelah itu, semua kode biner di antara biner terendah dan biner tertinggi digunakan untuk merepresentasikan bilangan riil antara R_{min} dan R_{max} dengan jarak interval yang didapatkan dengan rumus:

$$Interval = (R_{max} - \frac{R_{min}}{2^n})/(2^n - 1) \quad (2.1)$$

c) Pengkodean Riil

Pengkodean riil dapat mengatasi kelemahan yang terdapat pada pengkodean biner yaitu tidak membutuhkan waktu banyak dalam hal konversi dari biner ke riil maupun sebaliknya (Mahmudy, 2015).

d) Pengkodean Integer

Pengkodean integer adalah pengkodean dengan menggunakan bilangan cacah dimana bilangan yang sama boleh muncul lebih dari satu kali (Mahmudy, 2015).

e) Pengkodean Permutasi

Pengkodean permutasi mirip dengan pengkodean integer yakni dengan menggunakan bilangan cacah, namun bedanya pada pengkodean permutasi ini setiap bilangan hanya boleh muncul satu kali saja karena menunjukkan urutan (Mahmudy, 2015).

2.5.2 Skema Pemilihan *Parent*

Dalam GA, pemilihan *parent* didasarkan pada beberapa skema pilihan yang berbeda. Beberapa diantaranya akan dibahas di bawah ini (Bhatt & Chauhan, 2015) :

a) *Roulette wheel*:

Roulette wheel adalah strategi seleksi *parent* paling terkenal di GA. Pertama-tama, hitung nilai fungsi *fitness* dari semua kromosom. Misalkan F adalah jumlah total dari semua nilai *fitness* kromosom yang diharapkan. Hitung probabilitas

$$P_s = \frac{Nilai\ Fitness\ Kromosom}{F} \quad (2.2)$$

, kemudian hitung probabilitas secara kumulatif untuk semua kromosom. Bangkitkan angka acak R antara 0 dan 1, lalu pilih *parent* sesuai dengan angka R. Ulangi prosedur ini sebanyak *n* kali di mana *n* adalah jumlah individu.

b) Tournament approach:

Pemilihan *tournament* digunakan untuk menghindari efek konvergensi dini dalam algoritme genetika. Pendekatan ini menggunakan nilai relatif dari fungsi *fitness* sebagai kriteria untuk memilih individu dan tidak seperti *roulette wheel* yang didasarkan pada proporsi nilai *fitness*. Dalam proses seleksi ini ukuran turnamen harus ditentukan dan kemudian untuk setiap nilai *fitness* individu harus dihitung dan dibandingkan sehingga individu terbaik dapat dipilih.

c) Linear Ranking:

Dalam metode *linear ranking*, individu-individu dipilih sesuai dengan probabilitas seleksi yang besarnya tergantung dari peringkat mereka. Di sini peringkat pertama diberikan ke kromosom terburuk (memiliki nilai *fitness* terendah) dan peringkat terakhir N (sama dengan jumlah total individu) ditetapkan untuk individu terbaik. (Bhatt & Chauhan, 2015).

Mekanisme *linear rangking* sebagai berikut (Shukla, Pandey, & Mehrotra, 2015):

Probabilitas linear ranking adalah:

$$P_i = \frac{1}{N}\left(\eta^- + (\eta^+ - \eta^-)\ \frac{i-1}{N-1}\right); i \in \{1, \ldots, N\} \quad (2.3)$$

dengan syarat,

$\eta^+ = 2 - \eta^-$ dan $\eta^- \geq 0$

Dimana:

i = ranking ke-*i* (setelah diurutkan)

P_i = Probabilitas dipilihnya individu ke-*i*.

$\frac{\eta^-}{N}$ = Probabilitas dipilihnya individu terburuk.

$\frac{\eta^+}{N}$ = Probabilitas dipilihnya individu terbaik.

Perhatikan bahwa semua individu mendapatkan peringkat yang berbeda, probabilitas pilihan yang berbeda, bahkan jika mereka memiliki nilai *fitness* yang sama (Blickle & Thiele, 1996).

d) Rank weighted mating strategy:

Dalam metode perkawinan *rank weighted mating strategy*, kromosom disusun sesuai urutan nilai fungsi *fitness* mereka dari kecil ke besar. Probabilitas seleksi (pp_k) dikaitkan dengan kromosom rank *k* yang memberikan probabilitas lebih tinggi untuk terpilih kepada kromosom dengan nilai fungsi *fitness* yang rendah.

Probabilitas terpilih

$$(pp_k) = \frac{2(n-k+1)}{n(n+1)} \quad (2.4)$$

dimana *n* adalah total jumlah dari kromosom (Golberg, 1989; Kodaganallur, Sen, & Mitra, 2014).

2.5.3 *Crossover*

Setelah pemilihan orang tua (*parent*), langkah selanjutnya adalah kawin atau rekombinasi. Dalam GA kromosom *parent* direkombinasi menggunakan mekanisme yang disebut *crossover*. Operator genetik ini dipilih sedemikian rupa sehingga keturunan (*offspring*) yang terbentuk memiliki probabilitas kelangsungan hidup lebih besar daripada *parent*. Berikut beberapa contoh operator *crossover* yang dapat digunakan dalam penjadwalan produksi (Bhatt & Chauhan, 2015).

a) *Precedence preservative crossover operator* (PPX): Setelah memilih dua *parent* dari kromosom, *vektor biner* dengan panjang yang sama dengan *parent* yang mewakili urutan di mana operasi dipilih dari *parent*. *Offspring* dibentuk dengan menggambar operasi paling kiri (tanpa pengulangan) dari *parent* sesuai dengan vektor biner (Mattfeld & Bierwirth, 2004). Contoh:

Parent 1 = a b c d e

Parent 2 = c d f a e b

Vector biner = 121122

Offspring 1 = a c b f d e

b) ***Two cut point crossover*:** Operator ini adalah *crossover* paling sederhana yang melibatkan pemilihan dua titik. *Offspring* pertama diciptakan dengan menukar bagian tengah antara dua titik potong orang tua. *Offspring* lain diciptakan dengan menukar bagian selain bagian tengah dari kedua orang tua (De Giovanni & Pezzella, 2010). Ilustrasi dua operator titik potong adalah sebagai berikut:

Parent 1= 100|1000 |11 *Parent* 1= 100 | 1000| 11
Parent 2= 001 |1001 | 11 *Parent* 2 = 001 |1001 |11
Offspring 1= 100 1001 11 *Offspring* 1 = 001 1000 11
Offspring 2= 001 1000 11 *Offspring* 2 = 100 1001 11

c) ***Partially mapped crossover* (PMX):** Seperti *two cut point crossover*, untuk membuat kromosom baru untuk generasi berikutnya, dua titik *crossover* dipilih secara acak dan string antara titik-titik ini dipertukarkan untuk membentuk keturunan baru (Moon, Lee, & Bae, 2008). Contoh PMX adalah sebagai berikut:

Parent 1 = 1243234132411423
Parent 2 = 2341312413423142
Offspring 1 = 1243**312413**411423
Offspring 2 = 2341**234132**423142

Untuk membuatnya dalam bentuk yang dapat diterima (*legal*) sebuah mekanisme perbaikan (*repair mechanism)* digunakan seperti:

Offspring 1 = 1243**312413**41**2**423 dan,
Offspring 2 = 2341**234132**423141

d) ***Modified Partially-Mapped Crossover* (mPMX):** Operator mPMX adalah modifikasi dalam operator PMX untuk meningkatkan rata-rata kualitas hasil solusi. Operator ini merupakan prosedur empat langkah (Kim & Kim, 2002).

Langkah 1: Pilih dua titik potong
Parent 1 = 123 |546| 897
Parent 2 = 425 |873| 916

Langkah 2: Pertukarkan string antara dua titik potong
Offspring 1 = 123 **873** 897
Offspring 2 = 425 **546** 916

Langkah 3: Memperbaiki *offspring* untuk menghilangkan duplikasi. Untuk memperbaiki *Offspring* harus dilakukan sedemikian rupa sehingga 8 diganti dengan 5, 7 diganti dengan 4 dan 3 diganti dengan 6 pada anak pertama. Demikian pula 5 adalah diganti dengan 8, 4 adalah diganti dengan 7 dan 6 diganti dengan 3 di keturunan kedua.

Offspring 1 = 12**6** 873 **5**9**4**

Offspring 2 = **7**2**8** 546 91**3**

Langkah 4: Urutkan pekerjaan dalam *batch* sesuai dengan aturan SPT (*shortest processing time*). Apabila diasumsikan pengurutan pekerjaan secara *ascending* (dari kecil ke besar) mengikuti aturan SPT, maka:

Offspring 1 = 126 378 459

Offspring 2 = 278 456 139

e) ***Position based crossover***: Seperti namanya, operator *position based crossover* ini berdasarkan pada posisi gen yang dipilih dari *parent*. Gen dipilih berdasarkan kehadiran bit '1' pada string bitmap. String bitmap ini memiliki panjang yang sama dengan panjang *parent* dan memiliki nilai acak '0' atau '1'. Setelah string bitmap dibentuk, *offspring* pertama dibuat dengan memilih gen *parent* pertama sesuai dengan string bitmap dan kemudian melakukan pengecekan gen di *parent* kedua. Gen-gen yang tidak terpilih dari *parent* kedua disalin ke *offspring* pertama pada posisi yang sama seperti *parent* kedua dan gen yang terpilih dari *parent* pertama jika juga hadir dalam *parent* kedua dimasukkan ke dalam *offspring* pertama menurut urutan relatifnya pada *offspring* pertama. *offspring* kedua dibentuk dengan menukar posisi kedua *parent* dan diproses dengan mengikuti prosedur yang sama seperti yang dijelaskan di atas (Yu & Liang, 2001). Sebagai contoh:

Parent 1 = 123654	*Parent* 2 = 346125
Bitmap = 001011	Bitmap = 001011
Parent 2 = 346125	*Parent* 1 = 123654
Offspring 1 = 356124	*Offspring* 2 = 163254

f) ***Linear order crossover***: Dalam *linear order crossover*, sebuah substring dipilih dari *parent* pertama dan ditempatkan di lokasi yang sama dan dengan urutan yang sama (urutan gen) dalam *offspring* kosong. Jika gen dari substring yang dipilih juga ada dalam *parent* kedua, maka gen tersebut dihapus dan sisa gen disalin ke *offsprint* kosong dengan mempertahankan urutan *parent* kedua (Liaw, 2000).

Parent 1 = 4 | 7 3 2 | 8 5 1

Parent 2 = 3 6 4 2 1 8 5

Offspring = 6 | 7 3 2 | 4 1 8 5

g) ***Uniform order-based crossover***: Operator ini mengambil gen dari *parent* pertama di setiap posisi dengan probabilitas

$$pb1 = ff(p1)/[ff(p1) + ff(p2)] \quad (2.5)$$

dimana *ff(p1)* dan *ff(p2)* adalah nilai fungsi *fitness* dari *parent* pertama dan *parent* kedua. Posisi gen yang kosong pada gen *offspring* diisi dengan gen

parent kedua dengan urutan seperti yang muncul pada *parent* kedua. Operator menciptakan *offspring* kedua dengan menukar peran *parent* tetapi dengan probabilitas

$$pb2 = ff(p2)/[ff(p1) + ff(p2)] \quad (2.6)$$

, contoh penerapan ditunjukkan di bawah ini (Schaller & Valente, 2013):

Parent 1: A B D E C F

Parent 2: C B E A F D

Offspring 1: A C D E B F

Offspring 2: B D E A F C

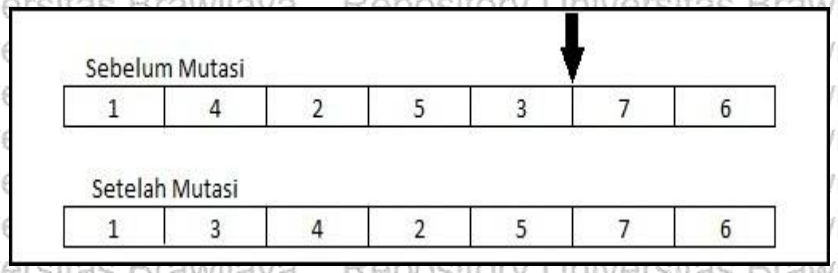
2.5.4 *Mutation*

Mutasi adalah operator genetik lain yang membantu GA untuk tidak terjebak dalam pengoptimalan lokal. Tampaknya menjadi operator penting dan harus diterapkan pada beberapa individu untuk membawa keragaman populasi. Operator mutasi yang berbeda telah disarankan dalam penelitian terdahulu. Beberapa contoh Operator mutase akan diilustrasikan di bawah ini (Bhatt & Chauhan, 2015).

a. ***Order based mutation***: Dua gen diambil secara acak dan ditukar selama proses mutasi (Cavory, Dupas, & Goncalves, 2005).
Misalnya, c1 adalah bentuk anak hasil dari *crossover*. Setelah mutasi c1 = 12**3**54**2**2 menjadi c1' = 12**2**54**3**2.

b. ***Refinement operator***: Operator ini menerapkan algoritme pencarian lokal untuk mengidentifikasi urutan yang berbeda dan menentukan urutan mana yang optimal. Menurut algoritme ini, pertama-tama harus diidentifikasi string paling *critical* yaitu string yang memiliki kontribusi besar dalam mengoptimalkan fungsi objektif dan kemudian operator pertukaran (*swapping*) diterapkan pada string yang dipilih. Proses ini diulang sampai didapatkan solusi yang lebih baik (De Giovanni & Pezzella, 2010).

c. ***Displacement mutation***: Algoritme ini memilih sebuah substring secara acak dan memasukkannya ke lokasi yang juga dipilih secara acak. Semua gerakan ini terjadi di dalam kromosom yang dipilih (Nearchou, 2004). Ilustrasi mutasi perpindahan ditunjukkan pada **Gambar 2.3**.



Gambar 2.3 Ilustrasi operasi *Displacement Mutation*

Sumber : (Bhatt & Chauhan, 2015)

d. ***Insertion mutation/ Inversion mutation***: *Insertion mutation* merupakan prosedur dua langkah. Langkah 1: pilih gen secara acak, Langkah 2: masukkan gen dari Langkah 1 ke lokasi yang dipilih secara acak (Zhang, Gao, & Shi, 2011). *Inversion mutation*, sebuah kromosom baru dibentuk dari dua poin yang dipilih secara acak dan kemudian membalikkan nilai gen di antara mereka (Dorndorf & Pesch, 1995; Falkenauer & Bouffouix, 1991).

e. ***Shift mutation***: Dalam *shift mutation*, gen dipilih secara acak dan kemudian digeser ke kiri atau kanan menuju ke posisi acak dari tempatnya sekarang (Bhatt & Chauhan, 2015).

2.5.5 Skema Pemilihan Generasi Berikutnya

Pada GA, tingkat eksplorasi memainkan peran penting untuk kecepatan konvergensi dan pendekatan kualitas. Selama proses seleksi, eksplorasi terjadi jika individu dengan kebugaran *sub-maximal* bertahan hidup. Di lain sisi, jika eksploitasi tinggi, pencarian akan memperoleh konvergensi sangat cepat tetapi akan terjebak dalam lokal optima. Jika eksplorasi tinggi, pencarian tidak akan terjebak dalam lokal optima tetapi kecepatan konvergensi akan jauh lebih rendah.

Keseimbangan antara memperbaiki individu yang baik dan memelihara yang menjanjikan harus ditemukan. Solusi yang baik seringkali merupakan kombinasi yang ditemukan pada individu yang kurang fit, oleh karena itu keragaman memainkan peran penting. Di sisi lain, seleksi harus memastikan kelangsungan hidup individu yang paling *fit* atau paling baik. Jika tidak, maka pencarian akan kurang lebih tidak terarah (Asteroth & Hagg, 2015). Beberapa strategi seleksi yang umum dipakai meliputi:

a. ***Fitness proportionate selection***, seperti pemilihan *roulette wheel* sebagaimana yang dijelaskan di atas.

b. ***Rank based selection***, seperti *linear ranking* atau *tournament* sebagaimana penjelasan di atas.

c. ***Elitism selection,*** seleksi elitism bekerja dengan mengumpulkan semua individu dalam populasi (*parent*) dan *offspring* dalam satu penampungan. Individu terbaik dalam penampungan ini akan lolos untuk masuk dalam generasi selanjutnya. Metode seleksi ini menjamin individu yang terbaik akan selalu lolos (Mahmudy, 2015).

d. ***Replacement Selection,*** Metode *replacement selection* memiliki dua aturan sebagai berikut: Pertama, keturunan (*offspring)* yang dihasilkan melalui proses mutasi akan menggantikan induknya (*parent*) apabila mempunyai nilai fitness yang lebih baik. Kedua, *offspring* yang dihasilkan melalui operasi *crossover* (dari dua *parent*) jika mempunyai nilai *fitness* yang lebih baik maka akan menggantikan induk yang terlemah dari keduanya. Metode *replacement selection* ini menjamin individu yang terbaik akan selalu lolos, tetapi tidak

menutup peluang bagi individu dengan nilai *fitness* rendah untuk lolos ke generasi berikutnya (Mahmudy, 2015).

2.5.6 *Stopping Criteria*

Dalam GA, proses komputasi akan berhenti apabila suatu kondisi berhenti telah terpenuhi. Beberapa kriteria yang biasa digunakan untuk menghentikan proses komputasi, antara lain:

a. Iterasi akan berhenti ketika generasi sampai pada generasi ke-*n*. Nilai *n* ini ditentukan di awal. Contoh penggunaan terdapat dalam (Wang & Luo, 2016).
b. Iterasi akan berhenti apabila sampai *n* generasi berurutan tidak didapatkan solusi yang lebih optimal (Mahmudy et al., 2012).
c. Iterasi akan berhenti ketika mencapai *t* satuan waktu. Cara ini biasa digunakan ketika dilakukan pembandingan performa antar metode (Mahmudy, 2014).

2.6 *Simulated Annealing*

Motivasi untuk algoritme *simulated annealing* (SA) berasal dari analogi antara proses *annealing* dan masalah optimasi kombinatorial. Proses *annealing* mengacu pada proses mencari status energi rendah dari suatu zat padat diawali dengan mencairkan substansi, dan kemudian menurunkan suhu perlahan, menghabiskan waktu yang lama pada suhu mendekati titik beku. Sebuah contoh akan menghasilkan kristal dari zat yang dicairkan. Dalam keadaan cair, partikel disusun secara acak. Apabila proses pendinginan sesuai dengan konfigurasi energi minimum maka dalam keadaan dasar yang padat akan memiliki struktur tertentu seperti yang terlihat dalam kristal. Jika pendinginan tidak dilakukan secara perlahan, padatan yang dihasilkan tidak akan mencapai keadaan dasar, tetapi akan membeku menjadi struktur lokal yang metastabil, seperti kaca atau kristal dengan beberapa cacat di dalamnya (Eglese, 1990).

Jadi dalam analogi tersebut, keadaan yang berbeda-beda dari substansi diibaratkan dengan kemungkinan solusi-solusi yang mungkin untuk masalah optimasi kombinatorial, dan energi dari sistem diibaratkan dengan fungsi yang harus diminimalkan (Kirkpatrick, Gelatt, & Vecchi, 1983). Berikut ini langkah-langkah penerapan algoritme *simulated annealing* untuk menyelesaikan suatu permasalahan:

1. Pembangkitan solusi awal

Untuk mencari solusi dari suatu permasalahan, pertama-tama akan dibangkitkan sebuah nilai x secara acak. Ketika membangkitkan solusi awal ini, nilai objektifnya juga harus ditentukan. Beberapa parameter lainnya yang juga perlu ditentukan di awal adalah iterasi maksimum, nilai objektif awal (E),temperatur awal (T_0), temperatur akhir (T_t), dan faktor reduksi suhu (α).

2. **Pencarian solusi baru**

Langkah selanjutnya yaitu pencarian solusi baru dengan cara membangkitkan nilai random serta menghitung nilai objektif dari solusi baru tersebut.

3. **Pengecekan parameter kontrol**

Pada tahapan ini, dilakukan pengecekan nilai parameter control untuk mengetahui perlu tidaknya parameter kontrol diturunkan. Apabila parameter kontrol perlu diturunkan, maka proses pencarian dilanjut ke langkah 4. Jika tidak diturunkan, maka proses diulang dari langkah 2 hingga langkah 3.

4. **Penentukan solusi baru yang terbaik**

Pada tahap ini akan didapatkan beberapa solusi baru, maka salah satu solusi terbaik harus dipilih untuk menggantikan solusi awal.

5. **Evaluasi solusi baru**

Tahap selanjutnya, dilakukan evaluasi untuk menentukan apakah solusi baru tersebut diterima atau tidak sebagai solusi global. Kriteria evalusai solusi ditunjukkan dengan rumus:

$$\Delta \mathrm{E} = \mathrm{E}(x_i + 1) - \mathrm{E}(x_i) \quad (2.7)$$

Keterangan:

ΔE = selisih nilai objektif

$\mathrm{E}(x_i + 1)$ = nilai objektif dari solusi baru

$\mathrm{E}(x_i)$ = nilai objektif dari solusi awal

Apabila nilai objektif solusi yang baru lebih kecil dari nilai objektif solusi awal ($\Delta \mathrm{E} \leq 0$), maka solusi baru diterima. Akan tetapi, apabila nilai objektif solusi baru lebih besar dari pada solusi awal ($\Delta \mathrm{E} > 0$), maka solusi baru masih dipertimbangkan dan memiliki probabilitas untuk terpilih dengan rumus:

$$P(\Delta \mathrm{E}) = e^{-\Delta \mathrm{E}/\mathrm{T}} > r \quad (2.8)$$

Keterangan:

T = Parameter kontrol (*temperature*)

r = Bilangan random antara 0 dan 1

6. **Penurunan parameter kontrol**

Setelah diperoleh solusi baru yang dapat diterima, maka parameter kontrol (*temperature*) diturunkan dengan menggunakan persamaan :

$$T_t = \alpha \times T_0 \quad (2.9)$$

Keterangan:

T_0 = Suhu awal

T_t = Suhu baru

α = Faktor reduksi suhu (α < 1)

7. Pengecekan iterasi maksimal

Penentuan batas maksimal iterasi digunakan untuk memutuskan apakah proses pencarian telah selesai atau tidak. Apabila telah selesai, maka solusi terbaik yang diperoleh merupakan solusi yang optimal dari permasalahan tersebut. Ulangi pencarian dari langkah 1 sampai langkah 7 apabila iterasi belum mencapai iterasi maksimal.

2.7 Hibridisasi Algoritme Metaheuristik

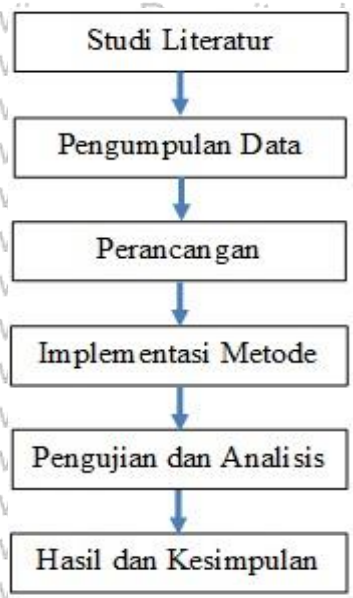
Hibridisasi metaheuristik digunakan di banyak karya ilmiah yang berfokus pada pemecahan masalah optimasi. Penggunaan teknik hibridisasi untuk pemecahan masalah optimasi semakin meningkat karena terbukti mampu memperoleh hasil yang baik. Hibridisasi metaheuristik didefinisikan sebagai kombinasi metaheuristik dengan metaheuristik lain dan / atau dengan metode lain yang biasanya berasal dari bidang komputasi cerdas dan riset operasi (Lopes Silva, de Souza, Freitas Souza, & de França Filho, 2018). Raidl menambahkan pada definisi hibridisasi, kemungkinan menggabungkan tidak hanya struktur metaheuristik lengkap tetapi juga komponen metaheuristik. Kombinasi ini bertujuan untuk memberikan sinergi antara metaheuristik, dari perbedaan karakteristik masing-masing algoritme, untuk memperoleh kinerja yang lebih baik bila dibandingkan dengan penggunaan setiap algoritme dalam bentuk murninya atau secara terpisah (Raidl, 2006).

Melihat hasil baik yang diperoleh dalam penerapan hibridisasi algoritme metaheuristik, maka pada penelitian ini akan digunakan hibridisasi metode *simulated annealing* dan *improved genetic algorithm* untuk menyelesaikan permasalahan penjadwalan produksi dan distribusi.

BAB 3 METODOLOGI

3.1 Metode Penelitian

Penelitian ini dilakukan dengan cara menerapkan metode yang diusulkan untuk menyelesaikan permasalahan penjadwalan produksi-distribusi mengacu pada permasalahan yang diusulkan dalam penelitian sebelumnya. Alur metode penelitian yang diajukan oleh penulis dapat dilihat pada **Gambar 3.1**.



Gambar 3.1 Alur Metode Penelitian

Berikut merupakan tahapan yang dilakukan penulis dalam melakukan penelitian ini:

1. Melakukan studi literatur yang berhubungan dengan permasalahan penjadwalan produksi-distribusi serta metode-metode penjadwalan lain yang dapat digunakan dalam menyelesaikan permasalahan tersebut.
2. Mengumpulkan data studi kasus dari perusahaan yang diajukan sebagai tempat penelitian.
3. Merancang model permasalahan penjadwalan produksi-distribusi ke dalam *mixed integer programming* (MIP).
4. Menerapkan metode yang diusulkan untuk menyelesaikan permasalahan produksi-distribusi yang telah diubah ke model MIP.
5. Menganalisis dan membandingkan hasil implementasi metode yang diusulkan dengan metode-metode pembanding lainnya.
6. Membandingkan dan menganalisa hasil implementasi metode yang diusulkan dengan metode penjadwalan produksi-distribusi nyata yang diterapkan pada perusahaan yang selanjutnya digunakan untuk menghitung selisih biaya

produksi-distribusi.

7. Membuat kesimpulan dari penerapan metode yang diusulkan dalam menyelesaikan permasalahan penjadwalan produksi-distribusi.

3.2 Studi Literatur

Beberapa studi literatur yang telah penulis lakukan antara lain: mencari permasalahan produksi-distribusi serupa dalam literatur sebelumnya, mengumpulkan berbagai dasar teori mengenai permasalahan penjadwalan produksi-distribusi, menganalisis penelitian-penelitian sebelumnya yang serupa dan menganalisis metode penjadwalan yang digunakan dalam penelitian sebelumnya untuk menyelesaikan permasalahan produksi-distribusi.

Dari studi literatur ini didapatkan beberapa parameter yang akan digunakan dalam penelitian ini, parameter-parameter tersebut dapat dilihat pada **Tabel 3.1** berikut.

Tabel 3.1 Parameter Dalam Penjadwalan

No.	Parameter	Penjelasan
1.	*Job*	Pesanan pelanggan yang akan diproduksi
2.	*Due date*	Tanggal jatuh tempo pengiriman
3.	*Volume*	Jumlah pesanan
4.	*Batch*	Tumpukan *job* yang diproses bersamaan
5.	*Capacity*	Kapasitas maksimal produksi per-*batch*
6.	*Set-up Cost*	Biaya set-up mesin *batching* satu kali produksi
7.	*Set-up Time*	Waktu set-up mesin *batching* satu kali produksi
8.	*Production Time*	Waktu proses produksi pada mesin *batching*
9.	*Delivery Cost*	Biaya pengantaran barang 1 kali perjalanan
10.	*Delivery Time*	Waktu pengantaran barang 1 kali perjalanan
11.	*Biaya Penalti*	Biaya penyimpanan barang oleh pelanggan
12.	*Total Cost*	Total biaya produksi

3.3 Data

Pada penelitian ini penulis menggunakan data yang dibangkitkan secara acak untuk melakukan pengujian metode yang diusulkan. Adapun aturan dalam membangkitkan data didasarkan pada penelitian Wang dan Lou (2016), aturan tersebut dapat dilihat pada **Tabel 3.2** dan **Tabel 3.3.**

1. Pembangkitan data secara acak dalam penjadwalan ukuran kecil

Dalam pengujian metode dengan menggunakan data berukuran kecil akan digunakan aturan pembangkitan data seperti yang terdapat pada **Tabel 3.2**

berikut.

Tabel 3.2 Aturan Pembangkitan Data Acak Berukuran Kecil

No.	Parameter	Penjelasan Ukuran
1.	*Job*	5, 7, 9, 11, 15
2.	*Due date*	Antara [0,2000]
3.	*Volume*	Antara [10,50]
5.	*Capacity*	Antara [50,100]
6.	*Set-up Cost*	Antara [200,500]
7.	*Set-up Time*	Antara [50,100]
8.	*Production Time*	Antara [1,5]
9.	*Delivery Cost*	Antara [300,500]
10.	*Delivery Time*	Antara [50,100]
11.	*Biaya Penalti*	Antara [1,5]

Sumber: Diadaptasi dari Wang dan Luo (2016)

2. Pembagkitan data secara acak dalam penjadwalan ukuran besar

Dalam pengujian metode dengan menggunakan data berukuran besar akan digunakan aturan pembangkitan data seperti yang terdapat pada **Tabel 3.3** berikut.

Tabel 3.3 Aturan Pembangkitan Data Acak Berukuran Besar

No.	Parameter	Penjelasan Ukuran
1.	*Job*	30, 50, 70, 100, 150
2.	*Due date*	Antara [0,2000]
3.	*Volume*	Antara [30, 50], [30, 100] dan [30, 150]
5.	*Capacity*	Antara [100,150] dan [150, 200]
6.	*Set-up Cost*	Antara [200,400]
7.	*Set-up Time*	Antara [10,40]
8.	*Production Time*	Antara [1,5]
9.	*Delivery Cost*	Antara [300,600]
10.	*Delivery Time*	Antara [50,100]
11.	*Biaya Penalti*	Antara [0.001, 0.005]

Sumber: Diadaptasi dari Wang dan Luo (2016)

3.4 Skenario Perancangan

Skenario Perancangan metode yang dilakukan oleh penulis yaitu:

1. Merepresentasikan parameter-parameter yang telah diperoleh dari studi

literatur ke dalam Model *Mix Integer Programming* dengan fungsi objektif untuk meminimalkan total biaya produksi. Bentuk sederhana dari model MIP ini dapat dilihat pada persamaan 3.1.

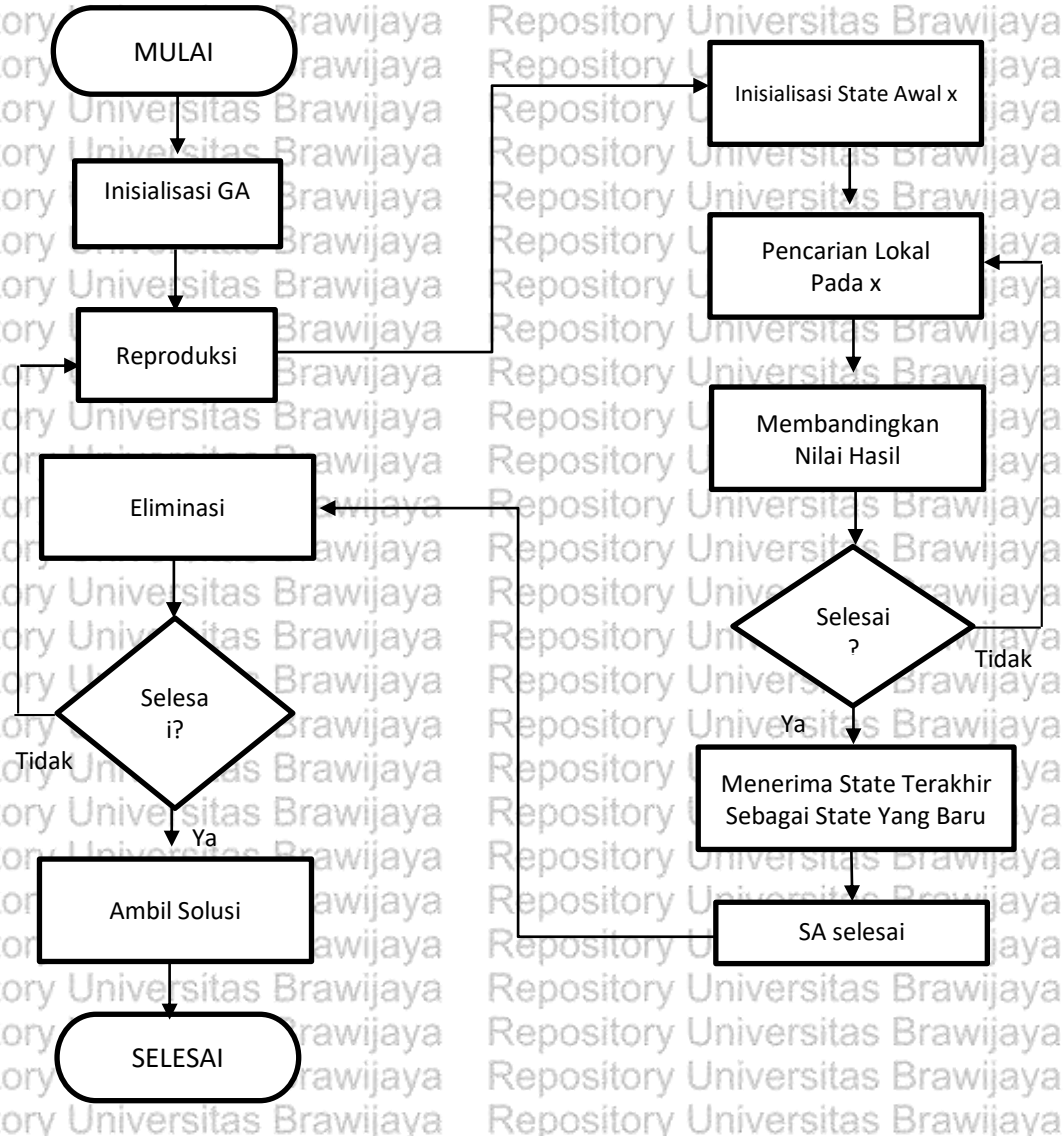
$$Min\ Z = S + D + P \quad (3.1)$$

Dimana:
Z = Total Biaya Produksi-Distribusi
S = Biaya Produksi
D = Biaya Distribusi
P = Pinalti

Selain persamaan 3.1 tersebut, juga terdapat batasan-batasan lain dalam penjadwalan yang harus dipenuhi.

2. Adapun skenario yang akan dilakukan dalam perancangan *improved genetic algorithm* yaitu dengan cara menguji parameter-parameter yang ada dalam algoritme genetika. Parameter-parameter yang akan diuji diantaranya: pemilihan teknik *mutation* dan *crossover*, penentuan metode seleksi dan eleminasi, penentuan *mutation rate* dan *crossover rate*, penambahan teknik lain (*improvisasi*) yang dapat diterapkan pada algoritme genetika sehingga mampu meningkatkan hasil pencarian solusi, dan juga pengujian jumlah populasi yang terbaik untuk menyelesaikan permasalahan model MIP.
3. Perancangan Hibridisasi metode *simulated annealing* dan *improved genetic algorithm* dilakukan dengan cara mengambil kelebihan metode *simulated annealing* untuk menutupi kekurangan metode algoritme genetika. Secara umum hibridisasi metode akan diterapkan seperti dalam poin-poin berikut.
   a. Algoritme genetika akan digunakan sebagai metode utama dalam hibridisasi metode ini. Oleh karena itu, semua tahapan dalam algoritme genetika akan dilakukan, seperti: membangkitkan generasi awal, pemilihan calon *parent* (seleksi), proses reproduksi (*mutation* dan *crossover*), eliminasi, dan perhitungan *fitness*.
   b. Pada proses *generate* solusi awal dan proses reproduksi pada algoritme genetika akan dilakukan sebuah improvisasi untuk meningkatkan performa dari algoritme genetika.
   c. Ditambahkannya operasi pencarian lokal (*local search*) tambahan pada sebagian individu setelah dilakukannya proses reproduksi sebagai sebuah *improvisasi*. Pada operasi ini akan diambil dua gen pada sebuah kromosom secara acak, kemudian kedua gen tersebut ditukar posisinya. Hasil pencarian solusi menggunakan pencarian lokal tambahan ini akan

dievaluasi menggunakan teknik dalam *simulated annealing*. Dimana, ketika operasi ini menghasilkan kromosom yang lebih baik dari kromosom sebelumnya maka akan menggantikan kromosom sebelumnya tersebut atau apabila hasil yang diperoleh lebih buruk dari kromosom sebelumnya tetap dapat menggantikan kromosom sebelumnya tersebut dengan probabilitas seperti dalam persamaan 2.8. Skenario hibridisasi metode *simulated annealing* dan *improved genetic algorithm* ini dapat dilihat pada **Gambar 3.2**



Gambar 3.2 Skenario Hibridisasi Metode *Simulated Annealing* dan *Genetic Algorithm*

3.5 Skenario Implementasi

Skenario dalam penerapan metode (implementasi metode) yang dilakukan oleh penulis yaitu dengan menerapkan hibridisasi metode yang diusulkan untuk

menyelesaikan model MIP. Berikut merupakan nilai-nilai yang akan dihitung dalam tahap implementasi ini.

a. Kasus Data Berukuran Kecil

Untuk kasus penjadwalan berukuran kecil, dari hasil penerapan hibridisasi metode ini akan diambil nilai hasil (*value*) dan waktu komputasi (*cpu time*). Selain itu, pada tahapan ini juga akan dihitung nilai hasil dan waktu komputasi dengan menggunakan metode pembanding *improved genetic algorithm*.

b. Kasus Data Berukuran Besar

Untuk kasus penjadwalan berukuran besar, hasil dari penerapan hibridisasi metode akan dihitung nilai *error* atau nilai deviasi (selisih antara total biaya menggunakan perhitungan *lower bound* dengan biaya yang dihasilkan metode) dan waktu komputasi (*average cpu time*) berdasarkan perhitungan *lower bound* yang diusulkan dalam (Wang, Grunder, et al., 2014). Nilai *error* (*ER*) didapatkan dengan rumus:

$$ER = (Heu - LB)/LB \quad (3.2)$$

Dimana:

Heu = Total biaya hasil penjadwalan menggunakan metode yang diusulkan.

LB = Total biaya hasil perhitungan menggunakan *lower bound.*

3.6 Skenario Pengujian

Pengujian akan dilakukan menggunakan PC laptop dengan RAM 4 GB, Windows 7 64-bit dan menggunakan Bahasa Pemrograman Java.

Adapun skenario yang akan dilakukan dalam pengujian metode hibridisasi adalah untuk mendapatkan nilai hasil (*value*), waktu komputasi (*cpu time*) dan nilai *error* maka akan dilakukan 2 (dua) tipe pengujian berdasarkan ukuran data yang diuji, yaitu:

1. Menggunakan Data Berukuran Kecil

Pengujian dilakukan sebanyak 5 kali percobaan (berdasarkan ukuran *job*) menggunakan data yang dibangkitkan mengacu pada **Tabel 3.2**. Dari setiap ukuran *job* tersebut akan dilakukan 10 kali pengujian kemudian akan diambil rata-rata nilai hasil dan rata-rata waktu komputasi. Dengan menggunakan data yang sama, pencarian rata-rata nilai hasil dan rata-rata waktu komputasi dihitung menggunakan metode pembanding yaitu *improved genetic algorithm*.

2. Menggunakan Data Berukuran Besar

Pada pengujian menggunakan data berukuran besar, akan dilakukan pengujian dengan dua skenario yaitu menggunakan kapasitas [100,150] dan [150, 200]. Untuk setiap skenario, kami mempertimbangkan tiga kasus berbeda dengan *volume* pekerjaan kecil, menengah dan besar secara berurutan dihasilkan secara acak menggunakan interval [30, 50], [30, 100] dan [30, 150]. pengujian dilakukan 30 kali percobaan (berdasarkan 5 ukuran *job*, 2 skenario dengan kapasitas berbeda, dan 3 *volume* yang berbeda) menggunakan data yang dibangkitkan mengacu pada **Tabel 3.3**. Dari setiap percobaan tersebut akan dilakukan 50 kali pengujian kemudian akan diambil rata-rata nilai *error* dan rata-rata waktu komputasi.

3.7 Skenario Pembanding

Untuk mengetahui penerapan metode yang diusulkan memperoleh hasil yang optimal, maka perlu dibandingkan dengan metode lainnya. Hasil dari penerapan metode yang diusulkan untuk menyelesaikan model MIP tersebut nantinya akan dibandingkan dengan hasil dari penerapan metode pembanding lain yakni *improved genetic algorithm* dan *lower bound* untuk diketahui rata-rata nilai hasil, rata-rata waktu komputasi dan rata-rata *error* atau nilai *deviasi*-nya. Kemudian hasil perbandingan metode yang diusulkan ini akan dibandingkan juga dengan hasil penelitian Wang dan Lou (2016).

BAB 4 PERANCANGAN

4.1 Perancangan Model Permasalahan

Dalam bab ini dibahas lebih detail mengenai parameter-parameter apa saja yang berhubungan dengan model permasalahan penjadwalan produksi-distribusi bermodel *make to order*. Kemudian akan diajukan satu model *mixed integer programming* (MIP) yang mencakup parameter-parameter permasalahan beserta kendala-kendala (*constrains*) yang ada. Adapun tujuan (*objective*) dari model MIP tersebut adalah mencari nilai biaya minimal produksi-distribusi.

Parameter-parameter yang digunakan dalam permasalahan ini dapat dilihat pada **Tabel 4.1** berikut.

Tabel 4.1 Deskripsi Parameter Permasalahan

Simbol	Deskripsi
$\mathbf{J}$	Set pekerjaan (*job*)
$\boldsymbol{n}$	Jumlah total *job* dalam 1 set pekerjaan
$\boldsymbol{i}$	Index urutan *job*
$\boldsymbol{j_i}$	*Job* ke-*i*
$\boldsymbol{d_i}$	Tanggal jatuh tempo *job* ke-*i*
$\boldsymbol{v_i}$	Jumlah *volume job* ke-*i*
$\boldsymbol{c}$	Kapasitas mesin untuk produksi satu tumpukan (*batch*)
$\boldsymbol{\lambda_c}$	Biaya set-up mesin *batching* satu kali produksi
$\boldsymbol{\lambda_t}$	Waktu set-up mesin *batching* satu kali produksi
$\boldsymbol{p_t}$	Waktu proses produksi pada mesin *batching*
$\boldsymbol{\eta_c}$	Biaya pengantaran barang 1 kali perjalanan
$\boldsymbol{\eta_t}$	Waktu pengantaran barang 1 kali perjalanan
$\boldsymbol{\beta_i}$	Biaya penyimpanan barang oleh pelanggan pada *job* ke-*i*
$\boldsymbol{u}$	Jumlah total tumpukan (*batch*)
$\boldsymbol{b}$	Index urutan tumpukan (*batch*)
$\boldsymbol{C_i}$	Waktu penyelesaian (kedatangan) untuk *job* ke-i
$\boldsymbol{C_b^B}$	Waktu penyelesaian (kedatangan) untuk tumpukan ke -*i*
$\boldsymbol{b_i}$	Index *batch* dimana *job* ke-*i* diproses
$\boldsymbol{Z}$	Total biaya produksi

Sumber: Wang dan Luo (2016)

Model MIP yang diajukan merupakan model permasalahan yang diadopsi dari penelitian (Wang & Luo, 2016). Berikut ini merupakan formulasi

permasalahan penjadwalan produksi-distribusi yang akan diselesaikan:

$$min\, Z = (\lambda_c + \eta_c).u + \beta_i.\{\sum_{i=1}^{n}(d_i - C_i)\} \quad (4.1)$$

Fungsi objektif 4.1 terikat pada kendala-kendala berikut:

1. Batasan 4.2 memastikan bahwa setiap *job* harus dikirimkan ke pelanggan sebelum atau pada saat tanggal jatuh tempo. Dalam penjadwalan ini keterlambatan tidak diperbolehkan.

$$C_i \leq d_i,\ \forall i \in [1,n], \quad (4.2)$$

2. Batasan 4.3 memastikan bahwa jumlah *job* dalam satu *batch* tidak melebihi kapasitas kendaraan.

$$\sum_{i=1}^{n} v_i \leq c,\ b_i = b, b \in [1,u], \quad (4.3)$$

3. Batasan 4.4 menunjukkan bahwa dua pekerjaan yang diproduksi dalam satu *batch* yang sama akan memiliki waktu penyelesaian yang sama.

$$C_b^B = C_i, \forall i \in \frac{[1,n]}{b_i} = b, b \in [1,u], \quad (4.4)$$

4. Batasan 4.5 mendefinisikan properti waktu penyelesaian dua *batch* berturut-turut. Properti waktu ini menunjukkan bahwa satu *batch* dapat diproses oleh mesin *batching* hanya setelah *batch* sebelumnya telah selesai sepenuhnya, dan satu *batch* dapat diangkut oleh kendaraan hanya setelah *batch* sebelumnya telah benar-benar dikirimkan.

$$C_b^B \leq C_{b+1}^B - max\{\lambda_t, \eta_t\}, b \epsilon [1,u], \quad (4.5)$$

Meskipun model MIP memberikan solusi optimal, variabel dan kendala meningkat secara drastis ketika jumlah *job* meningkat. Sebelum mengusulkan hibridisasi metode *simulated annealing* dan *improved genetic algorithm*, pertama-tama kami menunjukkan beberapa asumsi langsung ke model MIP.

(1) Terdapat jadwal optimal untuk masalah ini sehingga tidak ada waktu menganggur (*idle time*) antara pekerjaan pertama dan terakhir yang diproses di setiap *batch*.

(2) Terdapat jadwal yang optimal untuk masalah di mana keberangkatan setiap *batch* dilakukan segera pada waktu penyelesaian *batch* pada mesin *batching*.

4.2 Perancangan Metode *Genetic Algorithm*

Pada bagian ini dijelaskan perancangan algoritme genetika yang akan digunakan dalam penelitian ini. Perancangan algoritme ini terdiri dari skema

pembentukan representasi kromosom (*encoding*) dan skema penerjemahannya (*decoding*), pemilihan operator seleksi, pemilihan operator *crossover* dan mutation yang digunakan dalam proses reproduksi, pemilihan metode eleminasi dan perhitungan nilai *fitness*.

4.2.1 Rancangan penerapan algoritme genetika

Berikut ini dijelaskan langkah-langkah algoritme genetika yang diusulkan dalam tesis ini.

Langkah 1: Menginisialisasi probabilitas operator *crossover (cr*) dan operator mutasi *(mr*). Atur indeks pembangkitan, $g = 0$. Hasilkan sejumlah *popsize* kromosom sebagai populasi awal *pop (0)* dengan cara acak dan buatan. Kemudian, lakukan perbaikan operator (dijelaskan di bagian perbaikan operator) untuk setiap kromosom pada populasi yang baru dihasilkan *pop*(0).

Langkah 2: Jika kondisi terminasi terpenuhi, maka hentikan algoritme dan pilih solusi terbaik dalam populasi sebagai solusi akhir dari masalah.

Langkah 3: Evaluasi nilai *fitness* kromosom dalam populasi *pop*(*g*)

Langkah 4: Pilih kromosom untuk dijadikan *parent* pada proses reproduksi.

Langkah 5: Ulangi langkah-langkah berikut pada proses reproduksi *pop(g)*.

a) Dengan probabilitas cr, terapkan operator *crossover* ke dua kromosom induk dan menghasilkan dua keturunan. Lakukan perbaikan operator pada keturunan yang baru dihasilkan. Jika tidak ada operasi silang yang terjadi, maka hasilkan dua keturunan dengan menyalin persis kromosom *parent* mereka masing-masing.

b) Dengan probabilitas mr, hasilkan *offspring* baru dengan cara menyalin kromosom *parent* lalu terapkan operator mutasi, kemudian lakukan perbaikan operator pada keturunan yang baru dihasilkan.

Langkah 6: Lakukan operator eliminasi.

Langkah 7: Ubah $g = g + 1$. Ulangi langkah 2.

4.2.2 Skema pembentukan kromosom (*encoding*)

Untuk representasi kromosom, ada berbagai metode pengkodean yang sering digunakan seperti yang telah dibahas di bab sebelumnya. Pengkodean GA memiliki dampak penting pada kinerja algoritme seperti kemampuan pencarian dan keragaman populasi. Menggabungkan dengan karakteristik masalah penjadwalan dalam penelitian ini, penulis memilih metode pengkodean *Integer*. angka integer menunjukkan keputusan di tumpukan (*batch*) ke berapa sebuah pekerjaan dikerjakan. Sedangkan urutan gen (dari kiri ke kanan) dalam satu individu menunjukkan urutan *job* ke berapa dalam satu set *job*.

Sebagai contoh, untuk masalah dengan satu set pekerjaan dengan *10 job*, J = {1, 2, ..., 10}, struktur kromosom yang layak disajikan seperti yang ditunjukkan pada **Gambar 4.1**, di mana angka bilangan *integer* 3 di gen pertama menunjukkan pekerjaan ke-1 diproses pada *batch* ke 3 dan seterusnya.

Urutan Job Dari Kiri ke Kanan									
1	2	3	4	5	6	7	8	9	10
Representasi Kromosom Integer									
3	2	3	1	1	2	3	2	1	2

Gambar 4.1 Ilustrasi Representasi Kromosom

4.2.3 Skema penerjemahan kromosom (*decoding*)

Tujuan proses decoding adalah untuk menerjemahkan kromosom (*integer*) ke dalam jadwal lengkap yang dapat dievaluasi oleh fungsi *fitness*. Penjadwalan yang lengkap dari permasalahan ini harus mencakup informasi berikut: (1) Informasi *batch*, menjelaskan pekerjaan mana yang ditugaskan untuk *batch* mana, (2) urutan pemrosesan pekerjaan dan (3) waktu penyelesaian (*arrival time*) dari setiap pekerjaan ke pelanggan. Atas dasar pertimbangan di atas, akan digunakan dua tahapan untuk menangani proses decoding. Tahap pertama dari algoritme memperoleh informasi *batch* dan urutan pemrosesan *batch*, dan tahap kedua mendapatkan waktu kedatangan *job* ke pelanggan.

Tahap 1: mendapatkan informasi *batch* dan urutan pemrosesan *batch*.

Langkah 1. Inisialisasi indeks *job* ke nilai 1.

Langkah 2. Untuk setiap $j_i$secara berurutan sampai j_n.

Langkah 3. Tetapkan *job* j_i ke *batch* b_i.

Tahap 2: mendapatkan waktu penyelesaian setiap pekerjaan.

Langkah 1. Untuk setiap *batch b*, $1 \leq b \leq u$.

Langkah 2. Menginisialisasi waktu penyelesaian *batch b*, menjadi $C_b^B = \infty$,

Langkah 2.1. Untuk setiap *job i* di dalam *batch b*.

Langkah 2.2. Jika $C_b^B > d_i$ maka ubah $C_b^B = d_i$. Kembali ke Langkah 1.

Langkah 3. Untuk setiap *batch b* secara berurutan, $b = u, u-1, 2, \ldots, (u-1)$.

Langkah 4. Loop : Jika $C_{b-1}^B \leq C_b^B - n_t$, maka ubah $C_{b-1}^B = C_b^B - n_t$.

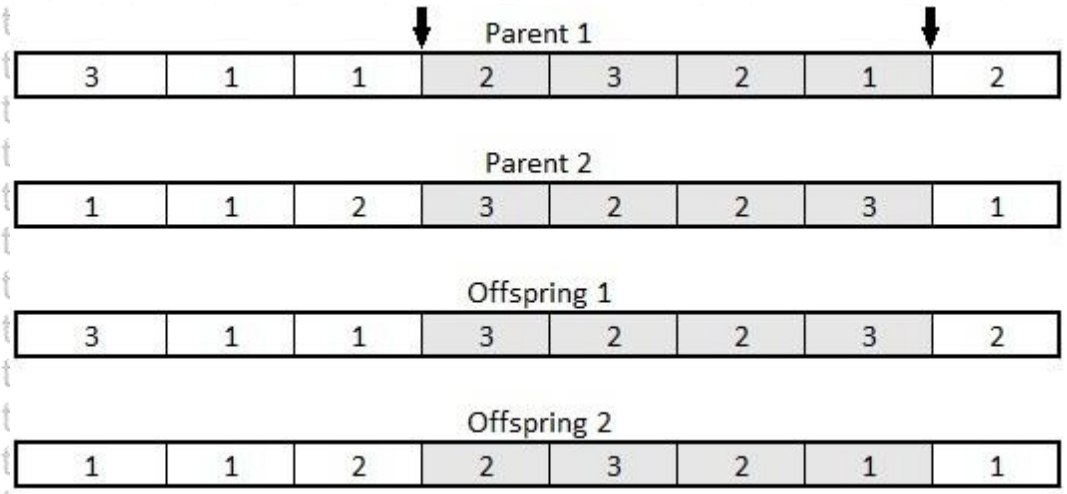
Contoh penerjemahan dan perhitungan kromosom (*decoding*) dapat dilihat pada sub-bab 4.5.

4.2.4 Pemilihan Jenis Operator Seleksi

Sebelum dilakukan proses reproduksi, calon *parent* diseleksi terlebih dahulu menggunakan metode *roulette wheel*. Diharapkan dengan menggunakan metode *roulette wheel* ini dapat memberi kesempatan kepada calon *parent* yang memiliki *nilai fitness* kecil untuk tetap terpilih dalam proses reproduksi. Hal ini dilakukan untuk menghindari lokal optimal.

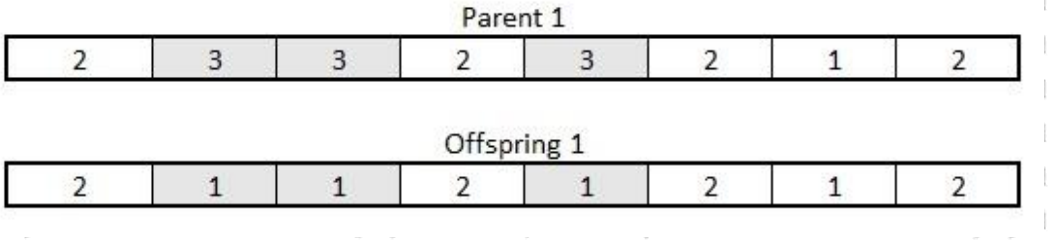
4.2.5 Pemilihan Jenis Operator Reproduksi

Pada proses reproduksi digunakan dua buah operator yaitu *crossover* dan *mutasi*. Jenis *crossover* yang dipilih adalah t*wo cut point crossover*. Operator ini adalah *crossover* paling sederhana yang melibatkan pemilihan dua titik. Hal pertama yang dilakukan untuk melakukan proses *two cut point crossover* adalah memilih dua titik *crossover*, kemudian menggunakan dua titik tersebut sebagai acuan pertukaran gen. Contoh penerapan *two cut point crossover* diilustrasikan seperti pada **Gambar 4.2**.



Gambar 4.2 Ilustrasi *Two Cut Point Crossover*

Mutasi digunakan untuk menghasilkan gangguan kecil pada kromosom untuk meningkatkan keragaman populasi. Ada beberapa operator mutasi yang telah dibahas di bab 2 sebelumnya seperti *inverse mutation*, *insertion mutation* dan *shift mutation*. Dalam penelitian ini, penulis akan menggunakan operator mutasi yang dapat dijelaskan sebagai berikut: Memilih secara acak gen yang merepresentasikan nomor *batch*, dan kemudian merubah nilainya secara acak dalam kisaran [1, u]. Agar lebih mudah dipahami, penerapan operator mutasi diilustrasi seperti pada **Gambar 4.3**.



Gambar 4.3 Ilustrasi operator mutasi

4.2.6 Pemilihan Jenis Operator Eliminasi

Operator eliminasi dalam algoritme genetika berfungsi untuk memilih individu-individu yang akan menjadi generasi selanjutnya. Jenis operator eliminasi yang digunakan dalam penelitian ini adalah *elitism selection*. Cara kerja *elitism selection* yaitu pertama-tama menghimpun semua individu dalam satu generasi baik *parent* maupun *offspring* dalam satu wadah. Kemudian, beberapa individu terbaik dalam wadah tersebut akan lolos untuk masuk dalam generasi selanjutnya. Metode seleksi ini memastikan hanya individu terbaik yang selalu lolos. *Elitism selection* dipilih dalam penelitian ini karena memiliki waktu komputasi yang cepat.

4.2.7 Perhitungan Nilai *Fitness* Dan *Stopping Criteria*

Nilai *fitness* adalah ukuran untuk menentukan seberapa baik sebuah solusi dibandingkan dengan fungsi objektif yang sebenarnya. Untuk masalah meminimalkan biaya, calon solusi dengan biaya lebih rendah mencerminkan solusi yang lebih baik, dan sebaliknya. Sehingga setiap kromosom *h* memiliki nilai *fitness* yang dapat dievaluasi dengan kebalikan dari fungsi objektif persamaan 4.1.

$$f_h = \frac{1}{(\lambda_c+\eta_c).u + \beta_i.\{\sum_{i=1}^{n}(d_i - c_i)\}} \quad (4.6)$$

Pada hibridisasi metode IGA dan SA, yang digunakan sebagai *stopping criteria* dalam adalah jumlah generasi.

4.2.8 Improved *Genetic Algorithm*

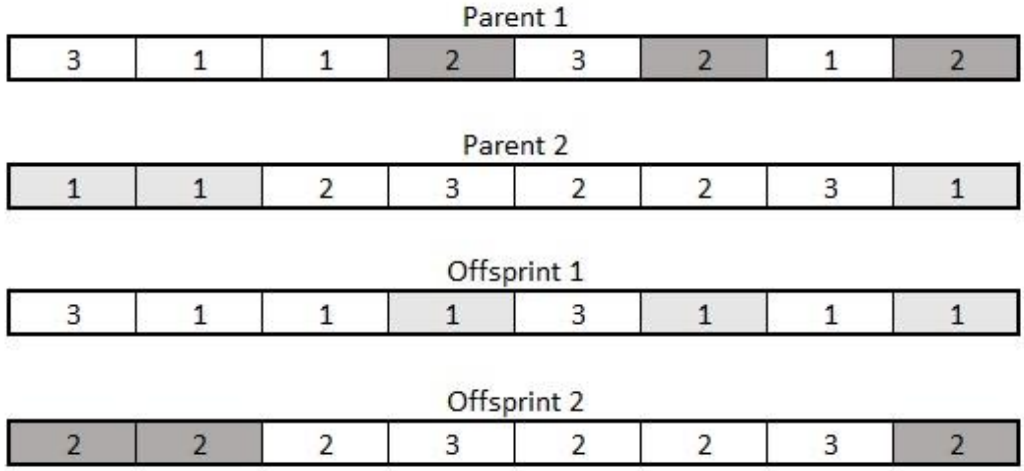
Pada operator-operator algoritme genetik berikut dilakukan perbaikan dengan harapan akan meningkatkan performa pencarian solusi.

a. Repair operator

Pada populasi awal atau populasi yang baru dihasilkan, setiap kromosom harus memenuhi batasan kapasitas. Jika kromosom tidak memenuhi batasan kapasitas, kromosom akan diperbaiki hingga memenuhi batasan kapasitas. Mekanisme perbaikan dari operator perbaikan ini dirancang untuk memindahkan pekerjaan dari *batch* yang memiliki kapasitas berlebih ke *batch* lain dengan kapasitas yang masih kurang.

b. *Crossover*

Memakai dua jenis *crossover* dengan probabilitas yang sama: Pertama, menggunakan t*wo cut point crossover*. Kedua, memilih acak dua index *batch b*, kemudian menukar posisi kedua index tersebut. Ilustrasi penggunaan *crossover* kedua ini dapat dilihat pada **Gambar 4.4**.



Gambar 4.4 Ilustrasi Penggunaan *Crossover* Kedua

c. *Mutation*

Menggunakan dua jenis mutasi dengan probabilitas yang sama: Pertama, menggunakan o*rder-based mutation*. Kedua, Memilih secara acak gen yang merepresentasikan nomor *batch*, dan kemudian merubah nilainya secara acak dalam kisaran [1, u] seperti ilustrasi pada **Gambar 4.3**.

d. *Generate populasi awal yang terarah*

Salah satu improvisasi yang diterapkan pada penelitian ini adalah dengan cara mengarahkan atau membatasi cakupan nilai acak pada kromosom awal. Batasan ini didapatkan dari perhitungan *lower bound* menggunakan persamaan 4.8. Hal ini dilakukan agar pencarian solusi akan lebih terarah sehingga diharapkan dapat meningkatkan efektifitas dari *genetic algorithm*. Jadi, nilai populasi awal didapatkan dari nilai acak dengan jangkauan [1, u] dimana nilai u merupakan hasil perhitungan *lower bound* dari persamaan 4.8.

4.3 Perancangan Hibridisasi Metode

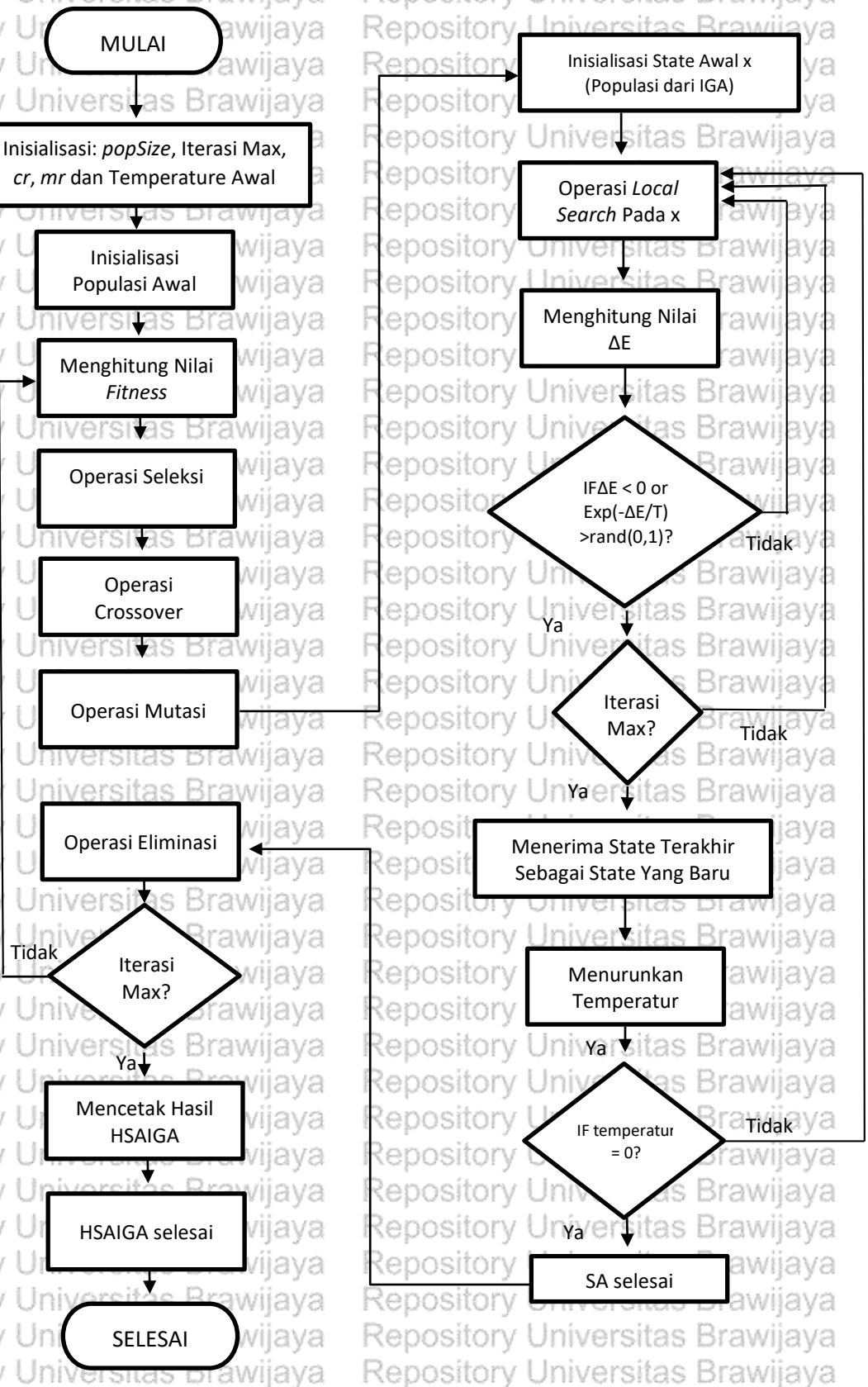
Perancangan hibridisasi metode *simulated annealing (SA)* dan *improved genetic algorithm* dilakukan dengan cara mengambil kelebihan metode *simulated annealing* untuk menutupi kekurangan metode algoritme genetika. Algoritme genetika akan digunakan sebagai metode utama dalam hibridisasi metode ini. Dalam hibridisasi metode ini, semua tahapan dalam algoritme genetika akan dilakukan, seperti: membangkitkan generasi awal, pemilihan calon *parent*

(seleksi), proses reproduksi (*mutation* dan *crossover*), eliminasi, perhitungan *fitness*. Langkah-langkah penerapan algoritme genetik dapat dilihat pada sub-bab 4.2. Sebagai tambahannya, pada penelitian ini akan diterapakan beberapa perbaikan (*improvement*) pada operator-operator algoritme genetika seperti yang telah dijelaskan pada sub-bab *4.2.8*.

Pada metode algoritme genetika, juga akan dilakukan operasi pencarian lokal (*local search*) tambahan pada sebagian individu setelah dilakukannya proses reproduksi sebagai sebuah perbaikan metode (*improvement*). Pada operasi ini akan diambil dua gen pada sebuah kromosom secara acak, kemudian kedua gen tersebut ditukar posisinya. Hasil pencarian solusi menggunakan pencarian lokal tambahan ini akan dievaluasi menggunakan teknik dalam *simulated annealing*. Dimana, ketika operasi ini menghasilkan kromosom yang lebih baik dibandingkan dengan kromosom sebelumnya maka akan menggantikan kromosom sebelumnya tersebut atau apabila hasil yang diperoleh lebih buruk dari kromosom sebelumnya tetap dapat menggantikan kromosom sebelumnya tersebut dengan probabilitas seperti dalam persamaan 2.8.

Perbedaan penelitian ini dengan penelitian sebelumnya (Wang & Luo, 2016) adalah adanya perbaikan (*improvement*) pada algoritme genetika yang selanjutnya diajukan menjadi metode Hibridisasi *simulated annealing* dan *improved genetic algorithm* (HSAIGA). Pada HSAIGA, SA dianggap sebagai operator tambahan dalam IGA yang digunakan dalam operasi pencarian lokal tambahan (*local search*). Diterapkannya metode SA sebagai operator pencarian lokal diharapkan dapat mengoptimalkan pencarian solusi.

Skenario hibridisasi metode *simulated annealing* dan *improved genetic algorithm* ini dapat dilihat pada **Gambar 4.5**



Gambar 4.5 Skenario Hibridisasi Metode *SA* dan *Improved GA*

4.4 Penggunaan Metode Pembanding

4.4.1 *Improved Genetic Algorithm*

Pada penelitian ini juga akan dibandingkan hasil pencarian solusi penjadwalan apabila hanya digunakan *improved genetic algorithm* tanpa ditambahkannya operasi *simulated annealing* didalamnya. Pada uji coba pembandingan kedua metode ini, nilai yang akan dibandingkan adalah rata-rata nilai total biaya dan rata-rata waktu komputasi. HSAIGA dan *improved genetic algorithm* akan dijalankan sebanyak 30 kali mengunakan kasus penjadwalan dengan *volume* dan waktu jatuh tempo yang sama. Parameter-parameter yang digunakan pada pengujian ini juga sama yaitu jumlah populasi = 50, jumlah generasi = 150, jumlah, kombinasi *mr* = 0.5 dan *cr* = 0.5.

4.4.2 *Lower Bound*

Dalam permasalahan penjadwalan ini, untuk mendapatkan solusi optimal dengan waktu komputasi yang wajar merupakan suatu hal yang sulit bahkan ketika permasalahan hanya memiliki 20 pekerjaan (*job*). Sehingga, di sini penulis akan menggunakan metode *Lower Bound* (*LB*) yang digunakan Wang, Grunder dan Moudni (2014) untuk mengevaluasi hibridisasi metode yang diusulkan. Wang, Grunder dan Moudni (2014) membagi persamaan 4.1 menjadi dua permasalahan *P1* dan *P2* kemudian mencari nilai $l1$ sebagai *lower bound* permasalahan *P1* dan $l2$ sebagai *lower bound* permasalahan *P2*. Hasil penjumlahan $l1$ dan $l2$ digunakan sebagai LB dalam permasalahan penelitian ini $LB = l1 + l2$. Berikut merupakan penjelasan *P1* dan *P2* serta cara mendapatkan $l1$ dan $l2$.

1. Permasalahan P1 berikut digunakan untuk mendapatkan $l1$. Dimisalkan tidak ada biaya penyimpanan pelanggan untuk setiap *job*, yaitu β = 0, maka masalah optimasi penjadwalan menjadi:

$$min\, Z = (\lambda_c + \eta_c) \times u \tag{4.7}$$

Dengan batasan kapasitas $\boldsymbol{c}$ seperti pada persamaan 4.3, yaitu:

$$\sum_{i=1}^{n} v_i \leq c,\ b_i = b, b \in [1, u],$$

Masalah optimasi dengan persamaan 4.7 merupakan permasalah *bin-packing*. Martello dan Toth (Martello & Toth, 1990) telah mengusulkan *Lower Bound* yang efisien untuk menyelesaikan permasalahan *bin-packing* ini. Berikut merupakan gambaran LB yang diusulkan.

Teorema 4.1: Diberikan contoh I dari masalah *bin-packing*, dan bilangan bulat apapun α, dimana $0 \leq \alpha \leq c/2$, biarkan

$$J_1 = \{ J \in N : v_j > c - \alpha \}$$
$$J_2 = \{ J \in N : c - \alpha \geq v_j > c/2 \}$$
$$J_1 = \{ J \in N : c/2 \leq v_j \leq \alpha \}$$

Sehingga persamaan

$$L(\alpha) = |J_1| + |J_2| + \max\left\{0, \left\lceil \frac{\sum_{j \epsilon J_3} v_j - (|J_1| \times c - \sum_{j \epsilon J_2} v_j)}{c} \right\rceil\right\} \quad (4.8)$$

merupakan LB dari Z(I).

Jadi, LB terbaik dapat ditemukan dengan persamaan berikut:

$$L_2 = \max\{L(\alpha): 0 \leq \alpha \leq c/2, \alpha \; integer\} \quad (4.9)$$

Martello dan Toth (1990) menunjukkan bahwa L_2 dapat ditentukan secara efisien dengan cara berikut.

Teorema 4.2: Misalkan V adalah himpunan semua nilai yang berbeda, $vi \geq c$ / 2. Kemudian,

$$L_2 = \begin{cases} n & if\ V = \emptyset \\ \max\{L(\alpha) : \alpha \in V\} & otherwise \end{cases} \quad (4.10)$$

2. Permasalahan *P2* berikut digunakan untuk mendapatkan *lower bound l2*. Wang, Grunder dan Moudni (2014) menggunakan model yang dipelajari (Baptiste, 2000) untuk mencari optimasi dengan model persamaan seperti pada persamaan 4.11 berikut:

$$Min\ Z = \beta_i . \{\sum_{i=1}^{n} (d_i - C_i)\} \quad (4.11)$$

Dengan batasan seperti pada persamaan 4.2, 4.4, dan 4.5.

$$C_i \leq d_i, \ \forall i \in [1, n],$$
$$C_b^B = C_i, \forall i \in \frac{[1,n]}{b_i} = b, b \in [1, u],$$
$$C_b^B \leq C_{b+1}^B - max\{\lambda_t, \eta_t\}, b \epsilon [1, u],$$

Wang, Grunder dan Moudni (2014) memperoleh LB dengan dua langkah pendekatan di mana yang pertama adalah mengasumsikan bahwa biaya pinalti adalah 0, dan yang kedua adalah untuk mencari LB untuk permasalahan P2. Berikut langkah-langkah yang digunakan untuk memperoleh LB untuk permasalahan P2:

1. Diasumsikan setiap *job* diproses pada satu *batch* secara terpisah.
2. Urutkan *job* berdasarkan deadline dari yang terkecil ke yang terbesar.

3. Hitung total *cost* dengan rumus $d_i - C_i$ dengan C_i didapatkan dari dua pilihan berikut :

a) Apabila $C_{b-1}^B > C_b^B - n_t$, jadikan $C_{b-1}^B = C_b^B - n_t$; gunakan nilai dari pilihan (a) untuk melakukan perhitungan (b).

b) Apabila selisih dari C_{b-1}^B yang menerapkan point (a) dikurangin C_{b-1}^B tanpa menerapkan point (a) lebih besar sama dengan C_b^B dikurangi C_{b-1}^B tanpa menerapkan point (a), jadikan $C_b^B = C_{b-1}^B$ tanpa menerapkan point (a).

Untuk kasus penjadwalan berukuran besar, hasil dari penerapan hibridisasi metode akan dihitung nilai *error* atau deviasi dan waktu komputasi (*average cpu time*) berdasarkan perhitungan *lower bound* yang diusulkan oleh Wang, Grunder dan Moudni (2014). Nilai *error* (*ER*) atau nilai deviasi didapatkan dengan rumus:

$$ER/Deviasi = (Heu - LB)/LB \quad (4.12)$$

Dimana:
Heu = Total Biaya hasil penjadwalan menggunakan metode HSAIGA.
LB = Total Biaya hasil perhitungan menggunakan *Lower Bound*.

4.5 Contoh Perhitungan Menggunakan HSAIGA

Berikut ini merupakan contoh manualisasi penjadwalan dengan menggunakan hibridisasi metode *simulated annealing* dan *improved genetic algorithm*. Contoh:

Terdapat 6 *job* J = {1,2,3,4,5,6}, *volume* (v) dan tanggal jatuh tempo (d) yang terkait dengan setiap *job* i, i = 1, 2 ..., 6, diasumsikan sebagai berikut:

v_1 = 50, v_2 = 30, v_3 = 25, v_4 = 40, v_5 =20 dan v_6 = 35,

d_1 = 1150, d_2 = 1210, d_3 = 1100, d_4 = 1250, d_5 =1500 dan d_6 = 1200,

β_1 = 5, β_2 = 5, β_3 = 5, β_4 = 5, β_5 =5 dan β_6 = 5,

P_t waktu produksi satu *batch* = 5, *set-up cost* λ_c = 500, *set-up time* λ_t = 50, waktu pengiriman pulang-pergi η_t = 100, biaya pengiriman pulang-pergi η_c = 300, kapasitas c = 100.

Pada pencarian solusi menggunakan HSAIGA digunakan langkah-langkah dalam algoritme genetika seperti yang telah dijelaskan pada sub-bab 4.2

Misalnya solusi yang dihasilkan oleh HSAIGA dalam bentuk kromosom seperti **Gambar 4.6**,

1	2	3	4	5	6
1	1	2	2	1	2

Gambar 4.6 Contoh Solusi Hasil HSAIGA

Maka mengacu pada skema *decoding* pada sub-bab 4.2.3 akan dilakukan dua tahapan sebagai berikut:

Tahap 1: Mendapatkan informasi *batch* dan urutan pemrosesan *batch*.

Dari contoh solusi yang dihasilkan oleh HSAIGA seperti pada **Gambar 4.6**, didapatkan informasi:

1. Jumlah *batch* u adalah 2.
2. *Job* 1, 2 dan 5 termasuk dalam *batch* 1.
3. *Job* 3, 4 dan 6 termasuk dalam *batch* 2.

Tahap 2: Mendapatkan waktu penyelesaian setiap pekerjaan.

Dari perulangan langkah 1 dan 2 pada tahap 2 didapatkan:

C_2^B= min $\{d_1, d_2, d_5\}$ = 1150

C_1^B= min $\{d_3, d_4, d_6\}$ = 1100

Dari perulangan langkah 3 dan 4 pada tahap 2 didapatkan:

C_2^B= 1150

$C_1^B > C_2^B - \eta_t$, jadikan $C_1^B = C_2^B - \eta_t$ = 1050.

Sehingga, waktu penyelesaian setiap *job* i adalah sebagai berikut (diurutkan berdasarkan waktu penyelensaiannya): $C_3 = C_4 = C_6$ = 1050, $C_1 = C_2 = C_5$ = 1150. Penyesuaian waktu penyelesaian setiap *batch* dilakukan untuk memenuhi batasan pada persamaan 4.5.

Setelah diketahui informasi *batch*, urutan pemrosesan *batch* serta waktu penyelesaian setiap *job* maka nilai-nilai tersebut dapat dimasukkan ke dalam rumus 4-1.

$$Z = (\lambda_c + \eta_c).u + \beta_i.\{\sum_{i=1}^{n}(d_i - C_i)\}$$

$$Z = (500 + 300).2 + 5.\{(1150 - 1050) + (1210 - 1050) + (1500 - 1050) + (1100 - 1050) + (1250 - 1050) + (1200 - 1050)\}$$

$$Z = 1600 + 5\,(100 + 160 + 450 + 50 + 200 + 150)$$

$$Z = 1600 + 5500 = \mathbf{7100}.$$

Dari perhitungan ini dapat disimpulkan bahwa penjadwalan menggunakan HSAIGA sudah dapat menyelesaikan masalah penjadwalan produksi dengan tidak ada keterlambatan pengiriman barang dan meminimalkan jumlah *batch*, akan tetapi solusi penjadwalan seharusnya masih dapat dioptimalkan dengan menempatkan *job* pada *batch* yang waktu penyelesaiannya C_i^B tidak terlalu jauh

dari waktu jatuh tempo *job* tersebut d_i, sehinggan dapat diperoleh nilai *earliness pinalty* yang lebih kecil.

4.6 Perancangan Uji Coba Dan Evaluasi

Di bagian perancangan uji coba ini akan dijelaskan skenario uji coba metode HSAIGA. Uji coba ini digunakan untuk memperoleh nilai parameter terbaik dalam metode HSAIGA guna memperoleh solusi yang optimal. Dalam perancangan ini semua parameter pada hibridisasi metode *simulated annealing* dan *improved genetic algorithm* akan diujicobakan. Parameter-parameter tersebut antara lain: ukuran populasi, jumlah generasi, kombinasi *crossover rate* dan *mutation rate*, nilai temperatur, nilai faktor reduksi, iterasi maksimal dan perbandingan metode.

4.6.1 Uji Coba Ukuran Populasi

Ukuran populasi dalam algoritme genetika merupakan parameter yang cukup berpengaruh dalam menghasilkan solusi yang optimal. Semakin besar ukuran populasi maka solusi yang dihasilkan pun akan semakin beragam. Meskipun begitu, apabila penentuan ukuran populasi terlalu besar maka akan mengakibatkan waktu komputasi yang lama. Sehingga pengujian paremeter untuk mencari ukuran populasi yang tepat dalam menghasilkan solusi optimal perlu dilakukan. Skenario uji coba ukuran populasi dapat pada **Tabel 4.2**. Dalam uji coba ukuran populasi ini diterapkan penambahan nilai dengan kelipatan 10. Berikut merupakan nilai parameter awal yang digunakan dalam uji coba ukuran populasi:

a. Ukuran Populasi : 10 – 100
b. Jumlah Generasi : 150
c. *Crossover rate* : 0,5
d. *Mutation Rate* : 0,5

Tabel 4.2 Skenario Uji Coba Ukuran Populasi

Ukuran Populasi	Nilai *Fitness* Uji coba ke-										Rata-rata
	1	2	3	4	5	6	7	8	9	10	
10											
20											
30											
40											
50											
60											
70											
80											
90											
100											

4.6.2 Uji Coba Jumlah Generasi

Setelah uji coba ukuran populasi dilakukan, langkah selanjutnya ialah melakukan uji coba untuk mencari parameter jumlah generasi. Pencarian parameter ini dilakukan untuk mengetahui nilai parameter jumlah generasi yang tepat dalam menghasilkan solusi optimal. Pada uji coba jumlah generasi ini akan digunakan penambahan jumlah generasi dengan kelipatan 30. Skenario uji coba ini dapat dilihat pada **Tabel 4.3**. Berikut merupakan nilai parameter awal yang digunakan dalam uji coba jumlah generasi:

a. Ukuran Populasi : Hasil uji coba ukuran populasi terbaik
b. Jumlah Generasi : 30 – 300
c. *Crossover rate* : 0,5
d. *Mutation Rate* : 0,5

Tabel 4.3 Skenario Uji Coba Jumlah Generasi

Jumlah Generasi	Nilai *Fitness* Uji coba ke-										Rata-rata
	1	2	3	4	5	6	7	8	9	10	
30											
60											
90											
120											
150											
180											
210											
240											
270											
300											

4.6.3 Uji Coba Kombinasi *Crossover Rate* dan *Mutation Rate*

Setelah pengujian parameter jumlah generasi, selanjutnya dilakukan uji coba untuk mencari nilai kombinasi *crossover rate* (*cr*) dan *mutation rate* (*mr*) yang yang tepat dalam menghasilkan solusi optimal. Uji coba yang dilakukan adalah dengan menggunakan cara mengkombinasikan nilai *cr* dan *mr* yang mana jumlah kedua nilai tersebut sama dengan 1 (*satu*). Skenario uji coba kombinasi cr dan mr dapat dilihat pada **Tabel 4.4**. Berikut merupakan nilai parameter awal yang digunakan dalam uji coba kombinasi *cr* dan *mr*:

a. Ukuran Populasi : Hasil uji coba ukuran populasi terbaik
b. Jumlah Generasi : Hasil uji coba jumlah generasi terbaik
c. *Crossover* Rate : 0,1 – 0,9
d. *Mutation Rate* : 0,1 – 0,9

Tabel 4.4 Skenario Uji Coba Kombinasi *Cr* dan *Mr*

Kombinasi		Nilai *Fitness* Uji coba ke-										Rata-rata
mr	*cr*	1	2	3	4	5	6	7	8	9	10	
0,1	0,9											
0,2	0,8											
0,3	0,7											
0,4	0,6											
0,5	0,5											
0,6	0,4											
0,7	0,3											
0,8	0,2											
0,9	0,1											

4.6.4 Uji Coba Jumlah Solusi Baru

Setelah melakukan uji coba pada parameter algoritme genetik, berikutnya akan dilakukan uji coba pada parameter algoritme *simulated annealing*. Parameter pertama yang diujicobakan adalah jumlah solusi baru yang akan dibangkitkan. Uji coba dilakukan untuk mengetahui jumlah optimal solusi baru yang perlu dibangkitkan. Dalam uji coba ini digunakan nilai uji antara 3-15 % dari populasi seperti ditunjukkan pada **Tabel 4.5**. Berikut merupakan nilai parameter awal yang digunakan dalam uji coba jumlah solusi baru:

a. Jumlah solusi baru : 3, 5, 7, 10, 15 %
b. Temperatur (T) : 125
c. Faktor reduksi (α) : 0,5
d. Iteraksi Maksimal : 5

Tabel 4.5 Skenario Uji Coba Jumlah Solusi Baru

Jumlah Solusi Baru (persentase)	Nilai *Fitness* Uji coba ke-										Rata-rata
	1	2	3	4	5	6	7	8	9	10	
3											
5											
7											
10											
15											

4.6.5 Uji Coba Nilai Temperatur

Parameter kedua yang diujicobakan adalah nilai temperatur. Uji coba ini dilakukan untuk mengetahui temperatur awal yang optimal dalam mempengaruhi solusi pencarian lokal (*local search)* yang dihasilkan. Pertambahan nilai

temperatur dalam uji coba ini adalah kelipatan 25. Skenario pengujian dapat dilihat pada **Tabel 4.6**. Berikut merupakan nilai parameter awal yang digunakan dalam uji coba nilai temperatur:

a. Jumlah solusi baru : Hasil uji coba jumlah solusi baru terbaik
b. Temperatur (T) : 50 – 250
c. Faktor reduksi (α) : 0,5
d. Iteraksi Maksimal : 5

Tabel 4.6 Skenario Uji Coba Nilai Temperatur

Nilai Temperatur	Nilai *Fitness* Uji coba ke-										Rata-rata
	1	2	3	4	5	6	7	8	9	10	
50											
75											
100											
125											
150											
175											
200											
225											
250											

4.6.6 Uji Coba Nilai Faktor Reduksi

Parameter kedua algoritme *simulated annealing* yang diujikan adalah nilai faktor reduksi. Pengujian ini dilakukan untuk mengetahui nilai α yang optimal dalam mempengaruhi solusi pencarian lokal (*local search)* yang dihasilkan. Dalam uji coba ini digunakan nilai mulai dari 0,1 hingga 0,9 seperti ditunjukkan pada **Tabel 4.7**. Berikut merupakan nilai parameter awal yang digunakan dalam uji coba nilai faktor reduksi:

a. Jumlah solusi baru : Hasil uji coba jumlah solusi baru terbaik
b. Temperatur (T) : Hasil uji coba nilai temperatur terbaik
c. Faktor reduksi (α) : 0,1 – 0,9
d. Iteraksi Maksimal : 5

Tabel 4.7 Skenario Uji Coba Nilai Faktor Reduksi

Faktor Reduksi	Nilai *Fitness* Uji coba ke-										Rata-rata
	1	2	3	4	5	6	7	8	9	10	
0,1											
0,2											
0,3											

Faktor Reduksi	Nilai *Fitness* Uji coba ke-										Rata-rata
	1	2	3	4	5	6	7	8	9	10	
0,4											
0,5											
0,6											
0.7											
0.8											
0.9											

4.6.7 Uji Coba Jumlah Iterasi Maksimal

Parameter terakhir yang diujicobakan adalah jumlah iterasi. Uji coba ini dilakukan untuk mengetahui seberapa banyak jumlah iterasi yang diperlukan dalam menghasilkan solusi pencarian lokal (*local search*) yang optimal. Uji coba jumlah iterasi maksimal dimulai dari iterasi 1 hingga 10. Skenario uji coba dapat dilihat pada **Tabel 4.8**. Berikut merupakan nilai parameter awal yang digunakan dalam uji coba jumlah iterasi maksimal:

a. Jumlah solusi baru : Hasil uji coba jumlah solusi baru terbaik
b. Temperatur (T) : Hasil uji coba nilai temperatur terbaik
c. Faktor reduksi (α) : Hasil uji coba nilai faktor reduksi terbaik
d. Iteraksi Maksimal : 1 – 10

Tabel 4.8 Skenario Uji Coba Jumlah Iterasi Maksimal

Iterasi Maksimal	Nilai *Fitness* Uji coba ke-										Rata-rata
	1	2	3	4	5	6	7	8	9	10	
1											
2											
3											
4											
5											
6											
7											
8											
9											
10											

4.6.8 Uji Coba Perbandingan Metode

Uji coba perbandingan metode dilakukan dengan cara membandingkan nilai hasil (*value*) dan waktu komputasi (*cpu time*) yang dihasilkan menggunakan metode HSAIGA dengan nilai hasil dan waktu komputasi yang dihasilkan apabila

menggunakan metode *improved genetic algorithm* saja. Setiap metode akan dieksekusi sebanyak 50 kali kemudian diambil nilai rata-rata total biaya (*avg. value*) dan rata-rata waktu komputasi (*avg. cpu time*) seperti ditunjukkan pada **Tabel 4.9** berikut.

Tabel 4.9 Skenario Uji Coba Perbandingan Metode

Metode	Nilai *Fitness* Uji coba ke-										Rata-rata	
											Avg. Value	*Avg. CPU Time*
	5	10	15	20	25	30	35	40	45	50		
Hibridisasi *Simulated Annealing* dan *Improved Genetic Algorithm* (HSAIGA)												
Improved Genetic Algorithm (IGA)												

BAB 5 HASIL DAN PEMBAHASAN

5.1 Hasil Uji Coba Parameter HSAIGA

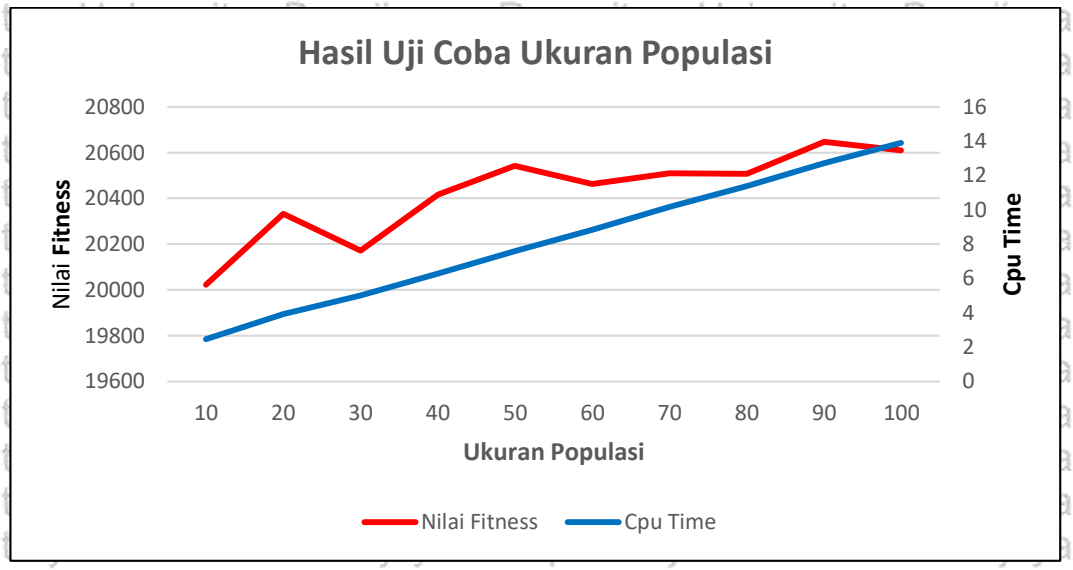
Pada bab ini akan dipaparkan hasil uji coba penerapan metode Hibridisasi *simulated annealing* dan *improved genetic algorithm* (HSAIGA) untuk menyelesaikan permasalahan penjadwalan produksi-distribusi dengan model *make to order*. Sebelum dilakukan pengujian metode HSAIGA untuk menyelesaikan permasalahan dalam penelitian ini, maka dilakukan pengujian untuk menentukan parameter-parameter terbaik yang akan digunakan. Hasil uji coba untuk mencari parameter terbaik untuk metode HSAIGA dijelaskan sebagai berikut.

5.1.1 Hasil Uji Coba Ukuran Populasi

Hasil uji coba parameter ukuran populasi pada metode HSAIGA dapat dilihat pada **Tabel 5.1**. Penulis menggunakan beberapa parameter yang telah ditentukan di awal yakni jumlah generasi = 150 serta kombinasi *cr* = 0,5 dan *mr* = 0,5 dalam melakukan uji coba ukuran populasi.

Tabel 5.1 Hasil Uji Coba Ukuran Populasi

Ukuran Populasi	Nilai *Fitness*										Rata-rata
	Uji coba ke-										
	1	2	3	4	5	6	7	8	9	10	
10	20157	20142	19727	20143	20161	19728	20139	20179	20088	19751	20021.5
20	20568	20609	20141	19770	20156	20575	20539	20172	19774	21017	20332.1
30	20165	20190	20166	20182	20161	20169	20123	19773	20591	20183	20170.3
40	20593	20581	20596	20606	20151	20158	20569	20570	20151	20174	20414.9
50	20195	20571	20167	20546	20606	20984	20590	20581	20180	20997	20541.7
60	20165	20582	20565	20596	20194	20179	21016	20176	20595	20569	20463.7
70	20591	20598	20604	20182	20590	20167	20599	20573	21034	20150	20508.8
80	20178	20610	20582	20164	20183	20599	20964	21018	20163	20608	20506.9
90	20201	20994	20613	20584	20616	20600	20600	20626	20619	21024	20647.7
100	20627	20595	20591	21025	20609	20612	21041	20212	20187	20584	20608.3



Gambar 5.1 Hasil Uji Coba Ukuran Populasi

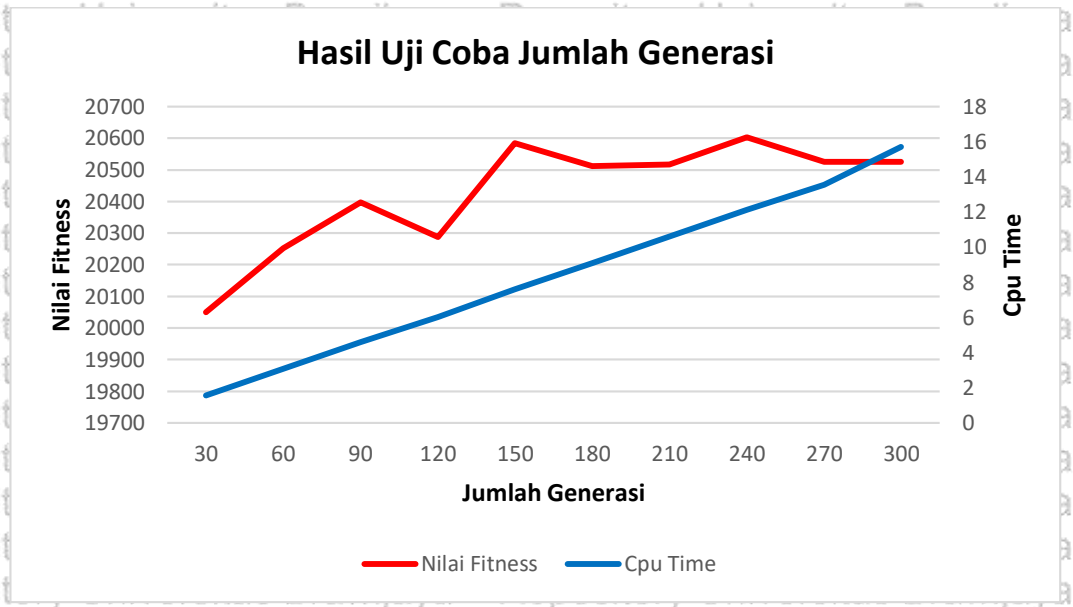
Apabila dilihat pada **Gambar 5.1**, titik terbaik terdapat pada ukuran populasi sebesar 50. Pemilihan parameter ukuran 50 diambil karena mempertimbangkan semakin besar ukuran populasi maka semakin besar pula waktu komputasinya. Walaupun dari **Gambar 5.1** dapat dilihat bahwa semakin besar ukuran populasi makan semakin baik juga nilai *fitness*-nya, akan tetapi tidak boleh disimpulkan bahwa semakin besar ukuran populasi maka nilai fitness-nya akan selalu terus meningkat. Hal tersebut dapat diamati pada saat ukuran populasi sebesar 90 ke ukuran 100 terjadi sedikit penurunan. Sedangkan dari ukuran populasi 50 ke ukuran populasi 100 tidak terdapat kenaikan nilai *fitness* yang terlalu besar. Mempertimbangkan hal-hal tersebut penulis memilih untuk menggunakan ukuran populasi 50 sebagai parameter pada metode HSAIGA ini.

5.1.2 Hasil Uji Coba Jumlah Generasi

Pada **Tabel 5.2** dapat dilihat hasil uji coba parameter jumlah generasi pada metode HSAIGA. Uji coba jumlah generasi dilakukan untuk mengetahui berapa jumlah generasi yang dapat diperlukan untuk memperoleh nilai optimal dan mengetahui terjadinya konvergensi. Konvergensi terjadi ketika nilai yang dihasilkan oleh generasi berikutnya tidak memperoleh kenaikan yang signifikan. Dalam melakukan uji coba untuk menentukan parameter jumlah generasi, digunakan beberapa parameter yaitu parameter ukuran populasinya = 50 yang didapat dari uji coba sebelumnya, parameter *cr* = 0,5 dan *mr* = 0,5 yang telah ditentukan di awal.

Tabel 5.2 Hasil Uji Coba Jumlah Generasi

Generasi	Nilai *Fitness*										Rata-rata
	Uji coba ke-										
	1	2	3	4	5	6	7	8	9	10	
30	20511	20104	20094	20102	20107	19690	19703	20038	20457	19699	20050.5
60	20124	19731	20533	20111	20137	20545	20110	20541	20541	20143	20251.6
90	20989	20150	20138	20573	20118	20571	20182	20541	20135	20567	20396.4
120	19771	20587	20584	19753	20127	20580	20164	20172	20564	20571	20287.3
150	20593	20588	20595	20591	20986	20156	21000	20585	20153	20588	20583.5
180	20185	20993	20983	20622	20592	20185	20196	20187	20589	20592	20512.4
210	20184	20176	20581	21020	21019	20609	20202	20188	20602	20594	20517.5
240	20605	20610	21028	21036	20569	20200	20571	20211	20580	20618	20602.8
270	21044	21055	20604	20181	20203	20167	20591	20595	20607	20208	20525.5
300	20558	20631	20616	20619	20201	20202	20598	20609	20599	20624	20525.7



Gambar 5.2 Hasil Uji Coba Jumlah Generasi

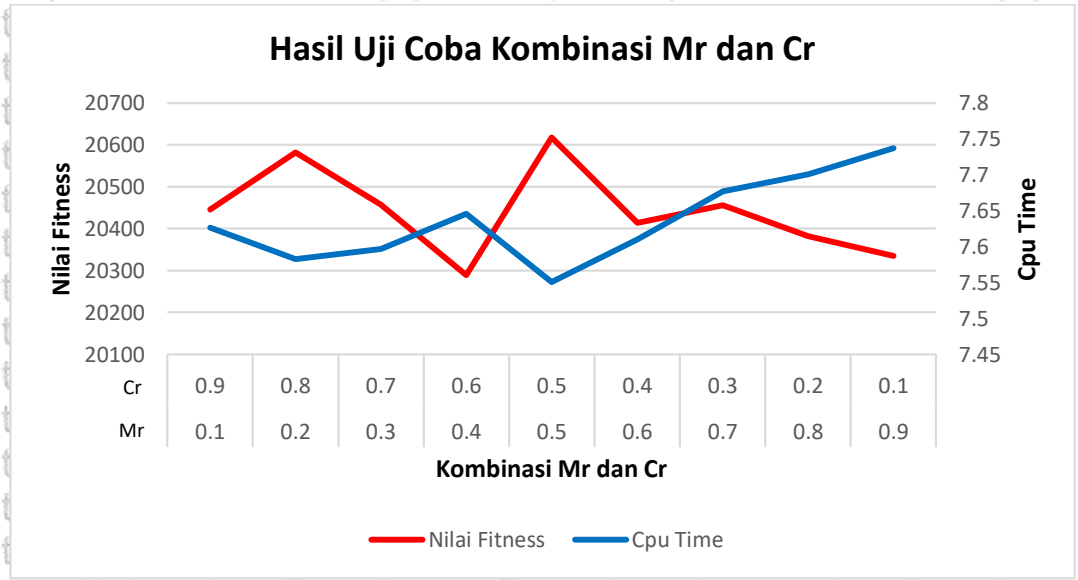
Pada **Gambar 5.2**, dapat dilihat titik terbaik terjadi pada saat generasi 150. Pada uji coba ini dapat dilihat bahwa semaik besar jumlah generasi maka semakin besar pula waktu komputasinya. Kenaikan nilai dari uji coba menggunakan 150 generasi sampai dengan uji coba menggunakan 300 generasi terjadi secara fluktuatif, akan tetapi nilai yang dihasilkan tidaklah terlalu berbeda jauh. Dari hasil uji jumlah generasi ini, maka dipilihlah jumlah generasi 150 sebagai parameter.

5.1.3 Hasil Uji Coba Kombinasi *Crossover Rate* dan *Mutation Rate*

Uji coba kombinasi *cr* dan *mr* dilakukan untuk mencari parameter kombinasi *cr* dan *mr* yang optimal. Setelah didapatkan parameter untuk ukuran populasi terbaik dan jumlah generasi terbaik yaitu sebesar 50 dan 150, maka parameter -parameter tersebut digunakan dalam pengujian parameter kombinasi *cr* dan *mr*. **Tabel 5.3** memuat hasil uji coba kombinasi *cr* dan *mr*.

Tabel 5.3 Hasil Uji Coba Kombinasi *Cr* dan *Mr*

Kombinasi		Nilai *Fitness*										Rata-rata
		Uji coba ke-										
mr	cr	1	2	3	4	5	6	7	8	9	10	
0.1	0.9	20580	20583	20584	20180	20548	20575	20547	20134	20561	20162	20445.4
0.2	0.8	20596	20187	20555	20595	20604	20578	20565	20567	20552	21019	20581.8
0.3	0.7	20553	21020	20149	20155	20596	20151	20169	20614	20592	20572	20457.1
0.4	0.6	20550	20587	20162	20168	20161	20606	20176	20163	20150	20165	20288.8
0.5	0.5	20539	20558	21003	20576	20594	20971	20606	20595	20152	20580	20617.4
0.6	0.4	20575	20595	20577	19763	20160	20578	20581	20177	20569	20569	20414.4
0.7	0.3	21016	20586	20593	20572	20151	20581	20593	20168	20186	20118	20456.4
0.8	0.2	20620	20169	20167	20576	20600	20166	20169	20191	20589	20577	20382.4
0.9	0.1	20579	19786	20163	20564	20588	20149	20578	20172	20590	20185	20335.4



Gambar 5.3 Hasil Uji Coba Kombinasi *Mr* dan *Cr*

Pada **Gambar 5.3**, dapat diketahui titik tertinggi dihasilkan pada saat kombinasi *cr* = 0,5 dan *mr* = 0,5. Selain itu, kombinasi tersebut juga memiliki rata-rata waktu komputasi terkecil. Pada uji coba ini juga dapat ditarik kesimpulan

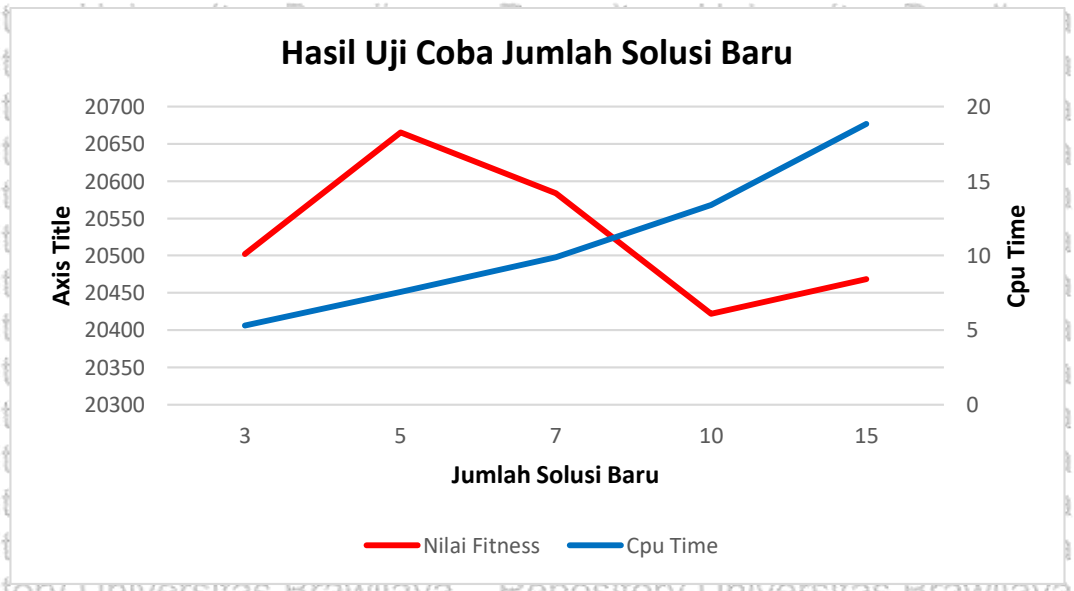
bahwa *crossover* dan mutasi memiliki peran yang seimbang dalam proses reproduksi untuk mendapatkan calon solusi yang baik. Hal tersebut juga menunjukkan bahwa *improvisasi* yang dilakukan penulis pada proses *crossover* dan *mutation* memberikan kemungkinan pencarian solusi yang lebih optimal sehingga menghasilkan nilai *fitness* yang tinggi.

5.1.4 Hasil Uji Coba Jumlah Solusi Baru

Hasil uji coba pemilihan parameter jumlah solusi baru ditunjukkan pada **Tabel 5.4**. Pada uji coba ini, solusi baru dibangkitkan dengan prosentase sebesar 3, 5, 7, 10 dan 15 % dari total populasi keseluruhan. Sebagai parameter awal uji coba jumlah solusi baru, makan penulis menggunakan beberapa parameter antara lain: temperatur = 125, faktor reduksi = 0,5 dan iterasi maksimal sebanyak 5.

Tabel 5.4 Hasil Uji Coba Jumlah Solusi Baru

Jumlah Solusi Baru (%)	Nilai *Fitness*										Rata-rata
	Uji coba ke-										
	1	2	3	4	5	6	7	8	9	10	
3	20560	20160	21020	20173	20570	20591	20189	20581	20594	20581	20501.9
5	20611	20561	20533	21024	21019	20574	20594	20581	20556	20599	20665.2
7	21017	20578	20567	20565	20185	20595	20582	20597	20564	20591	20584.1
10	20585	20575	20180	20599	20578	20155	20206	20580	20182	20582	20422.2
15	20184	20559	21013	20152	20170	20598	20201	20601	21023	20183	20468.4



Gambar 5.4 Hasil Uji Coba Jumlah Solusi Baru

Dari grafik pada **Gambar 5.4** dapat dilihat bahwa titik terbaik dicapai pada saat jumlah solusi baru yang dibangkitkan sebanyak 5 % dari total populasi.

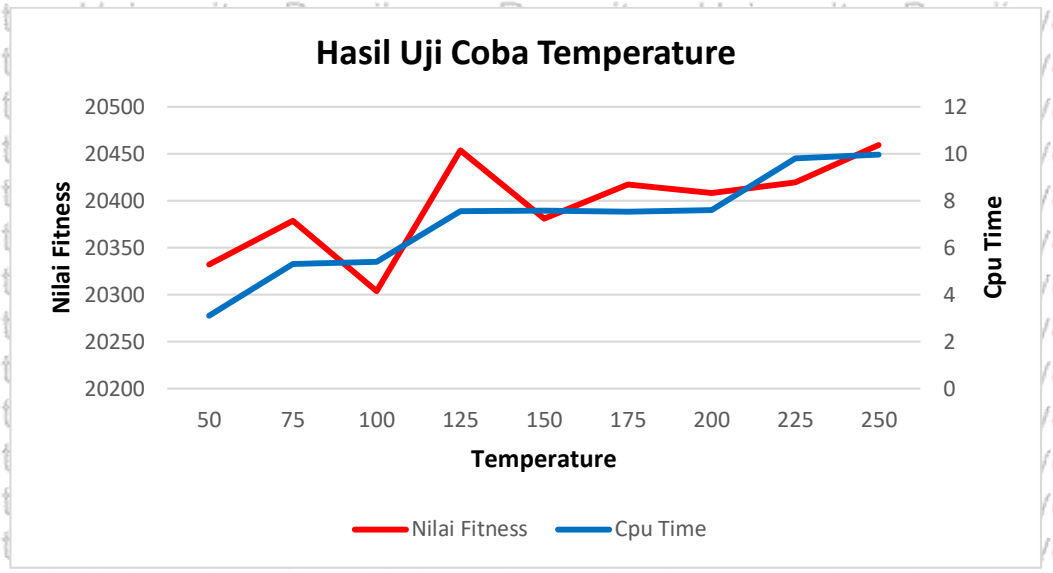
Mengingkatnya nilai parameter jumlah calon solusi baru tidak menjamin meningkatnya nilai kualitas solusi, akan tetapi sudah pasti menambah waktu komputasi. Hal tersebut terlihat pada uji coba dengan menggunakan lebih dari 5% total populasi dimana terjadi penurunan nilai yang cukup signifikan. Dari hasil uji coba ini, maka parameter jumlah solusi baru yang akan digunakan yaitu 5% dari total populasi.

5.1.5 Hasil Uji Coba Nilai Temperatur

Hasil uji coba pencarian parameter nilai temperatur dapat dilihat pada **Tabel 5.5**. Pada uji coba ini digunakan nilai temperature dengan kelipatan 25 mulai dari 50 sampai dengan 250. Parameter awal yang digunakan dalam pengujian nilai temperature antara lain: jumlah solusi baru 5%, faktor reduksi = 0,5 dan iterasi maksimal sebanyak 5. Sedangkan jumlah solusi baru yang dibangkitkan sebesar 5% (dari total populasi) yang diperoleh dari hasil uji coba parameter terbaik sebelumnya.

Tabel 5.5 Hasil Uji Coba Nilai Temperatur

Temp	Nilai *Fitness*										Rata-rata
	Uji coba ke-										
	1	2	3	4	5	6	7	8	9	10	
50	20154	20176	20580	20195	20169	20170	20955	20150	20178	20595	20332.2
75	20591	20203	20140	20167	20602	20159	20158	20195	20582	20987	20378.4
100	20165	20160	20172	20588	20584	20195	20177	19770	20203	21019	20303.3
125	20603	20577	20192	20987	20152	20529	20584	20579	20182	20150	20453.5
150	20178	20614	20185	20582	20179	20155	20585	20176	20568	20587	20380.9
175	20186	20999	20168	20161	20572	20573	20162	20592	20168	20592	20417.3
200	20553	20159	20173	20165	20580	20554	20582	20167	20556	20591	20408
225	20154	20600	20578	20178	21026	20178	20580	20182	20170	20551	20419.7
250	20555	20597	20559	20567	20190	20163	20600	20597	20164	20601	20459.3



Gambar 5.5 Hasil Uji Coba Temperature

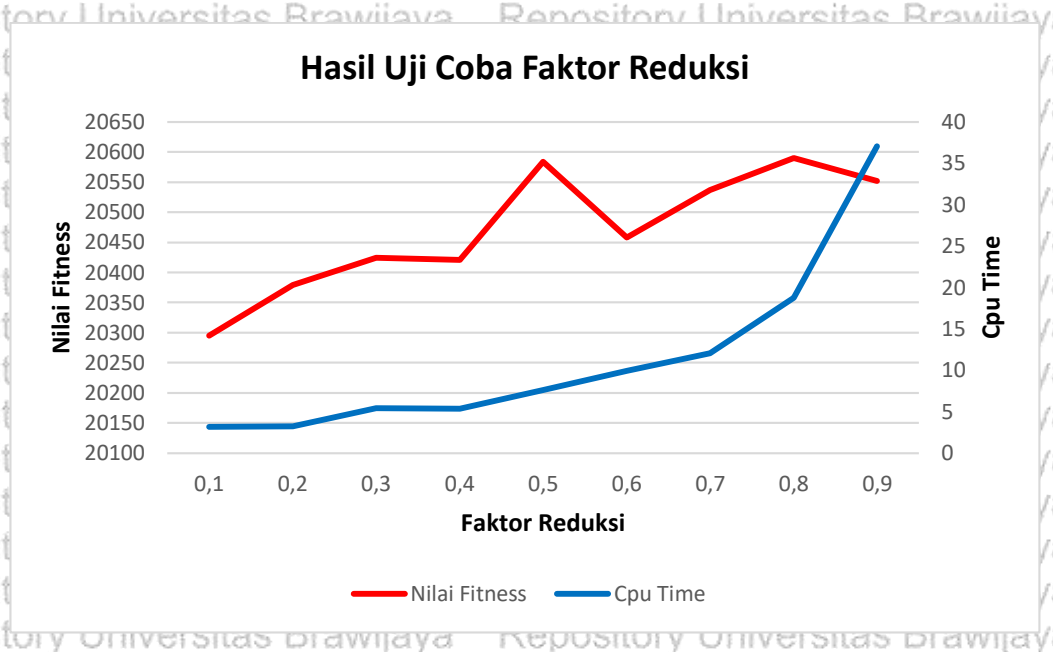
Dari **Gambar 5.5** dapat diketahui bahwa titik terbaik diperoleh pada temperatur 125. Semakin besar nilai temperature maka akan semakin mempengaruhi pertimbangan dalam pemilihan solusi baru. Walaupun demikian, dari hasil uji coba diatas dapat diketahui bahwa dari nilai temperature 125 ke atas tidak terjadi peningkatan nilai *fitness* yang terlalu signifikan. Jadi, nilai temperature 125 digunakan sebagai parameter dalam uji coba selanjutnya.

5.1.6 Hasil Uji Coba Nilai Faktor Reduksi

Hasil uji coba pencarian parameter nilai faktor reduksi dapat dilihat pada **Tabel 5.6**. Dalam uji coba ini, nilai faktor reduksi yang diujikan dimulai dari 0,1 hingga 0,9. Parameter lain yang digunakan dalam uji coba nilai temperatur yaitu jumlah solusi baru 5%, temperature awal 125 dan jumlah iterasi maksimal 5.

Tabel 5.6 Hasil Uji Coba Nilai Faktor Reduksi

Faktor Reduksi	Nilai *Fitness*										Rata-rata
	Uji coba ke-										
	1	2	3	4	5	6	7	8	9	10	
0,1	20592	20557	20187	20183	20183	20187	20173	20564	20147	20180	20295.3
0,2	20591	20575	20567	20187	20160	20612	20562	20193	19770	20579	20379.6
0,3	20173	20585	20568	20578	20165	20179	21032	20180	20595	20192	20424.7
0,4	20171	20577	20172	20573	20193	20568	20180	20588	20610	20578	20421
0,5	20596	20184	21000	20166	20568	20592	20577	20584	20153	20584	20500.4
0,6	20173	20196	20587	20185	20585	20181	20562	20574	20550	20573	20416.6
0,7	20598	20555	20553	20566	20622	20591	20581	20577	20564	20999	20620.6
0,8	20598	20586	20165	20625	20580	21000	20576	20605	20582	20580	20589.7
0,9	20603	20603	20578	20611	20602	20587	20185	20575	20594	20581	20551.9



Gambar 5.6 Hasil Uji Coba Faktor Reduksi

Berdasarkan hasil pengujian faktor reduksi pada **Gambar 5.6**, nilai parameter faktor reduksi terbaik yang diperoleh adalah 0,5. Nilai faktor reduksi digunakan untuk mengurangi nilai temperatur dimana nilai faktor reduksi ini juga digunakan ketika mempertimbangkan pemilihan solusi baru. Nilai faktor reduksi 0,5 yang diperoleh dari uji coba ini dipakai dalam uji coba parameter berikutnya.

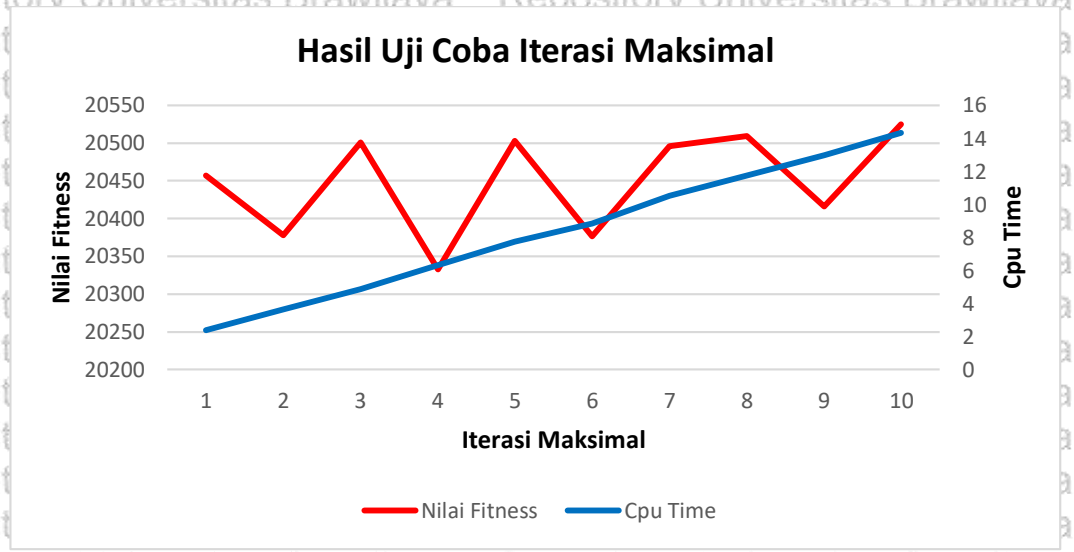
5.1.7 Hasil Uji Coba Jumlah Iterasi Maksimal

Hasil uji coba pencarian nilai parameter jumlah iterasi maksimal dapat dilihat pada **Tabel 5.7**. Nilai parameter yang diujicobakan untuk mencari jumlah iterasi maksimal mulai dari 1 hingga 10 kali iterasi. Beberapa parameter awal yang telah diperoleh sebelumnya yaitu jumlah solusi baru = 5% dari total populasi, temperature = 125, nilai faktor reduksi = 0,5 digunakan juga pada uji coba ini.

Tabel 5.7 Hasil Uji Coba Jumlah Iterasi Maksimal

Iterasi Max	Nilai *Fitness*										Rata-rata
	Uji coba ke-										
	1	2	3	4	5	6	7	8	9	10	
1	20176	20572	21009	20151	20175	20170	20589	20576	20568	20586	20457.2
2	20172	20185	20546	20577	20579	20197	20589	20161	20605	20169	20378
3	20165	20570	20174	21014	20574	20177	20593	20997	20174	20574	20501.2
4	20193	20599	20570	20555	20603	20145	20163	20170	20171	20160	20332.9
5	20598	20583	20569	20610	20580	20564	20163	20567	20206	20589	20502.9
6	20564	20162	20573	20591	20594	19756	20150	20182	20590	20602	20376.4
7	20581	20169	20552	20599	20579	20606	20582	20564	20582	20148	20496.2

Iterasi Max	Nilai *Fitness*										Rata-rata
	Uji coba ke-										
	1	2	3	4	5	6	7	8	9	10	
8	20582	20579	20599	20183	20593	20578	20600	20170	20180	21026	20509
9	20584	20146	20594	20182	20146	20584	20185	20540	20599	20601	20416.1
10	20180	20593	20584	20144	20587	20563	20445	20551	21025	20577	20524.9



Gambar 5.7 Hasil Uji Coba Iterasi Maksimal

Berdasarkan **Gambar 5.7** dapat dilihat bahwa nilai *fitness* dari r jumlah iterasi maksimal berubah secara fluktuatif. Dari grafik **Gambar 5.7** juga dapat ditarik kesimpulan bahwa banyaknya jumlah iterasi belum tentu menaikkan nilai *fitness*, karena ada kemungkinan solusi mengalami konvergensi dini. Pada uji coba ini dipilih jumlah iterasi maksimal = 5, dengan pertimbangan untuk memberi kesempatan metode *simulated annealing* untuk memperbaiki solusinya dan dengan waktu komputasi yang relatif cepat dibandingkan jumlah iterasi maksimal diatasnya.

5.2 Hasil Uji Coba Metode HSAIGA

Hasil penerapan metode yang diusulkan yaitu Hibridisasi *simulated annealing* dan *improved genetic algorithm* (HSAIGA) dalam menyelesaikan permasalahan model *mixed interger programming* yang merepresentasikan permasalahan penjadwalan distribusi model *make to order* ditunjukkan pada **Tabel 5.8** dan **Tabel 5.9**. Dari uji coba sebelumnya didapatkan semua nilai parameter terbaik yang dibutuhkan untuk digunakan pada metode HSAIGA. Adapun parameter-parameter terbaik yang telah didapatkan dari uji coba sebelumnya baik parameter pada *genetic algorithm* maupun parameter pada *simulated annealing* yaitu : Ukuran populasi = 50, jumlah generasi = 150,

kombinasi *mr* = 0.5 dan *cr* = 0.5, jumlah solusi baru = 5% dari total populasi, temperature = 125, faktor reduksi = 0.5, dan iterasi maksimal = 5. Tahap pengujian dilakukan dengan dua skenario yaitu menggunakan kapasitas [100,150] dan [150, 200]. Untuk setiap skenario, dipertimbangkan tiga kasus berbeda dengan *volume* pekerjaan kecil, menengah dan besar secara berurutan yang dihasilkan secara acak menggunakan interval [30, 50], [30, 100] dan [30, 150]. Setiap pengujian dilakukan percobaan berdasarkan 5 ukuran *job*, 2 skenario dengan kapasitas berbeda, dan 3 *volume* yang berbeda menggunakan data yang dibangkitkan mengacu pada **Tabel 3.3**. Dari setiap percobaan tersebut dilakukan 50 kali pengujian, kemudian akan diambil rata-rata nilai *error* dan rata-rata waktu komputasi. **Tabel 5.8** merupakan hasil pengujian metode HSAIGA pada kasus dimana kapasitas *batch* antara 100 sampai 150, sedangkan **Tabel 5.9** merupakan hasil pengujian metode HSAIGA pada kasus dimana kapasitas *batch* antara 150-200.

Tabel 5.8 Hasil Pengujian Metode HSAIGA Kapasitas 100-150

$c \in [100, 150]$						
Size (*n*)	$v_i \in [30, 50]$		$v_i \in [30, 100]$		$v_i \in [30, 150]$	
	A.ER(%)	A.CpuT(s)	A.ER(%)	A.CpuT(s)	A.ER(%)	A.CpuT(s)
30	7.065993	0.35928	4.629421	0.802133	2.453135	1.153233
50	7.919581	0.7885	6.136312	2.129675	4.93813	2.122633
70	9.00402	1.37914	6.603796	3.583625	6.069315	3.938656
100	10.31265	2.57802	9.756359	6.78615	7.196196	4.429925
150	10.93885	5.59708	10.77646	9.058606	8.923595	9.084161

Tabel 5.9 Hasil Pengujian Metode HSAIGA Kapasitas 150-200

$c \in [150, 200]$						
Size (*n*)	$v_i \in [30, 50]$		$v_i \in [30, 100]$		$v_i \in [30, 150]$	
	A.ER(%)	A.CpuT(s)	A.ER(%)	A.CpuT(s)	A.ER(%)	A.CpuT(s)
30	3.225541	0.605075	6.239231	0.389725	3.647786	0.472175
50	5.224994	0.6082	7.613364	0.86705	6.246264	1.160025
70	6.182237	1.059775	7.996217	1.518	8.581655	2.00755
100	6.075633	2.069347	9.300306	3.05645	10.19862	4.06255
150	8.379695	4.274886	9.737914	6.00705	10.61058	8.672159

Dari **Tabel 5.8** dan **Tabel 5.9** dapat dilihat bahwa rata-rata nilai *error* keseluruhan tidak lebih dari 11%. Hal ini membuktikan bahwa penerapat Hibridisasi s*imulated annealing* dan *improved genetic algorithm* memperoleh hasil yang lebih baik dibandingkan dengan hasil yang diperoleh pada penelitian sebelumnya oleh Wang dan Lou (2016) yang mempunyai rata-rata *error* tidak

lebih dari 16%.

5.3 Perbandingan Hasil Uji Coba Dengan IGA

Untuk mengetahui bahwa hibridisasi *simulated annealing* dan *improved genetic algorithm* dapat meningkatkan akurasi penjadwalan penjadwalan produksi-distribusi model *make to order*, maka perlu dibuktikan dengan cara membandingkannya dengan metode *improved genetic algorithm* tanpa ditambakan metode *simulated annealing* didalamnya. Pada **Tabel 5.10**, HSAIGA dan *improved genetic algorithm* dijalankan sebanyak 10 kali mengunakan kasus penjadwalan dengan *volume* dan waktu jatuh tempo yang sama. Selain itu, parameter-parameter yang digunakan pada pengujian ini juga sama yaitu jumlah populasi = 50, jumlah generasi = 150, jumlah, kombinasi *mr* = 0.5 dan *cr* = 0.5. Nilai yang dibandingkan pada **Tabel 5.10** ini adalah rata-rata nilai total biaya produksi dan rata-rata waktu komputasi kedua metode.

Tabel 5.10 Perbandingan Hasil Uji Coba HSAIGA Dan IGA

Size (n)	HSAIGA		IGA	
	A.Value (*Total Cost*)	A.CpuT(*s*)	A.Value (*Total Cost*)	A.CpuT(*s*)
5	2507.9	0.1818	2633.4	0.0248
7	3211	0.1877	3314	0.0315
9	3666.1	0.1859	3860	0.0327
11	4379.3	0.2376	4587.4	0.039
15	5782.1	0.331	6067.8	0.0514

Tabel 5.11 dan **Tabel 5.12** merupakan hasil pengujian metode IGA. Seperti tahap pengujian yang diterapkan pada HSAIGA, tahap pengujian IGA dilakukan dengan dua skenario yaitu menggunakan kapasitas [100,150] dan [150, 200]. Untuk setiap skenario, dipertimbangkan tiga kasus berbeda dengan *volume* pekerjaan kecil, menengah dan besar secara berurutan yang dihasilkan secara acak menggunakan interval [30, 50], [30, 100] dan [30, 150]. Setiap pengujian dilakukan percobaan berdasarkan 5 ukuran *job*, 2 skenario dengan kapasitas berbeda, dan 3 *volume* yang berbeda menggunakan data yang dibangkitkan mengacu pada **Tabel 3.3**. Dari setiap percobaan tersebut dilakukan 30 kali pengujian, kemudian akan diambil rata-rata nilai *error* dan rata-rata waktu komputasi.

Tabel 5.11 Hasil Pengujian IGA Kapasitas 100-150

$c \in [100, 150]$						
Size (*n*)	$v_i \in [30, 50]$		$v_i \in [30, 100]$		$v_i \in [30, 150]$	
	A.ER(%)	A.CpuT(s)	A.ER(%)	A.CpuT(s)	A.ER(%)	A.CpuT(s)
30	7.875094	0.1481	4.796192	0.23185	2.746474	0.2437
50	8.379979	0.31516	6.618377	0.4562	4.995706	0.51005
70	9.55229	0.52272	7.867106	0.83725	5.819478	0.8763
100	10.98285	0.8968	8.388709	1.64415	7.34351	1.68015
150	11.31312	1.90415	12.16007	3.1549	8.864132	3.5184

Tabel 5.12 Hasil Pengujian IGA Kapasitas 150-200

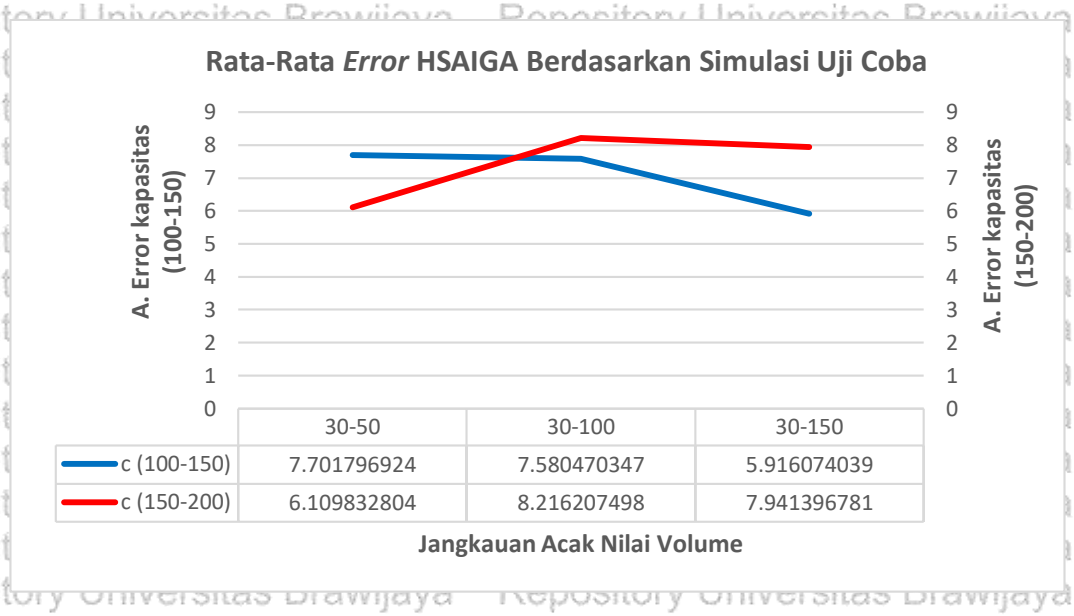
$c \in [150, 200]$						
Size (*n*)	$v_i \in [30, 50]$		$v_i \in [30, 100]$		$v_i \in [30, 150]$	
	A.ER(%)	A.CpuT(s)	A.ER(%)	A.CpuT(s)	A.ER(%)	A.CpuT(s)
30	3.340392	0.1485	6.266334	0.19685	3.459729	0.22505
50	5.299143	0.29045	7.665736	0.3885	7.622173	0.45385
70	6.487108	0.4733	8.779694	0.6446	8.827145	0.8095
100	6.316449	0.7908	9.324888	1.0966	11.39091	1.4296
150	9.340181	1.63685	10.52697	2.3882	11.89398	3.1547

Hasil pengujian IGA pada **Tabel 5.10** menunjukkan bahwa metode HSAIGA memiliki nilai rata-rata total biaya produksi (*total cost*) yang lebih kecil dari nilai rata-rata yang diperoleh menggunakan metode IGA. Dari **Tabel 5.11** dan **Tabel 5.12** dapat diketahui bahwa nilai rata-rata *error* yang diperoleh menggunakan metode IGA tidak lebih dari 13%. Hasil yang diperoleh menggunakan metode *improved genetic algorithm* ini sudah lebih baik bila dibandingkan dengan hasil yang diperoleh pada penelitian Wang dan Lou (2016). Dari hasil pengujian pada **Tabel 5.11** dan **Tabel 5.12**, dapat diambil kesimpulan bahwa secara keseluruan penerapan hibridisasi *simulated annealing* dan *improved genetic algoritm* dapat mengurangi rata-rata *error* pada penjadwalan dari nilai *error* yang diperoleh menggunakan IGA (lebih kecil dari 13%) menjadi nilai rata-rata *error* yang diperoleh dari HSAIGA (lebih kecil dari 11%).

5.4 Pembahasan Hasil Uji Coba HSAIGA

5.4.1 Hasil HSAIGA berdasarkan Simulasi Uji Coba

Pada sub-bab ini akan dibahas mengenai hasil pengujian HSAIGA untuk menyelesaikan permasalahan penjadwalan produksi-distribusi model *make to order* dimana hasil tersebut dapat dilihat pada **Tabel 5.8** dan **Tabel 5.9**. Apabila diambil nilai rata-rata *error* dari kedua tabel tersebut berdasarkan simulasi uji coba maka akan diperoleh nilai rata-rata *error* seperti pada **Gambar 5.8**.



Gambar 5.8 Rata-rata *Error* HSAIGA Berdasarkan Simulasi Uji Coba

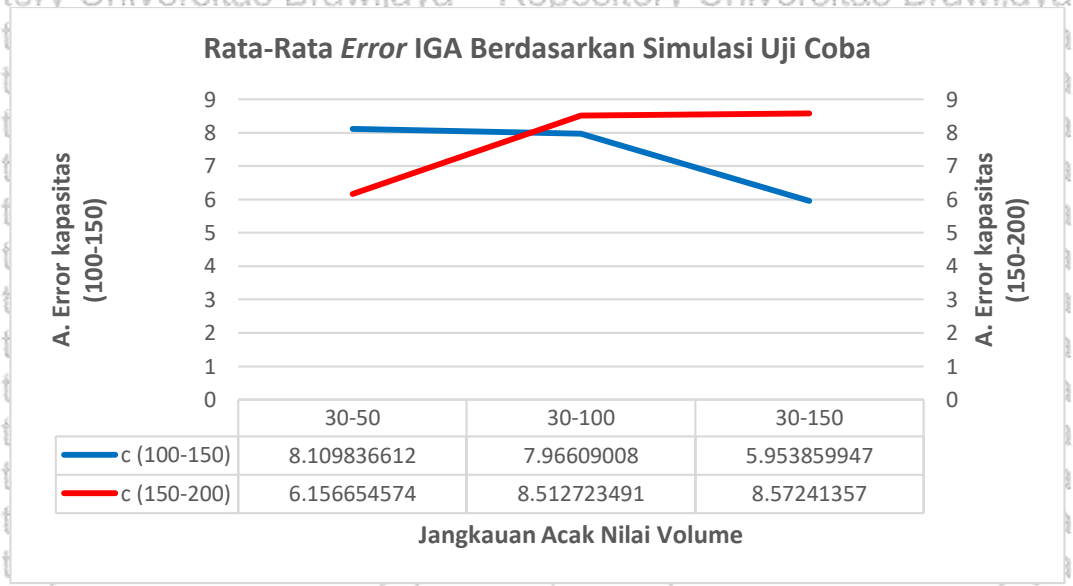
Dari **Tabel 5.8** dan **Tabel 5.9**, dapat dilihat bahwa nilai rata-rata *error* dan waktu komputasi meningkat seiring dengan meningkatnya ukuran pesanan (*n*). selain ukuran pesanan, ukuran *volume* dan kapasistas *batch* juga mempengaruhi nilai rata-rata *error*. Hal ini terlihat pada **Gambar 5.8**, pada kasus dimana kapasitas *batch* anatara 100-150 terjadi penurunan nilai rata-rata *error* seiring dengan bertambahnya besaran *volume* yang lebih mendekati kapasitas *batch*. Penurunan nilai rata-rata *error* ini terjadi dikarenakan semakin besar ukuran *volume* yang mendekati kapasitas *batch* maka akan semakin sedikit ruang yang tersisa (*space*) dalam satu *batch* untuk ditambahkan pesanan lainnya sehingga dapat mengurangi kesalahan penjadwalan dan menurunkan rata-rata nilai *error*.

Hal yang berbeda terjadi pada contoh kasus dengan kapasitas *batch* 150-200 dimana terjadi kenaikan nilai rata-rata *error* pada contoh kasus dengan *volume* 30-50 memiliki nilai rata-rata *error* terkecil dibandingkan nilai rata-rata *error* pada kasus yang lain. Dengan bertambahnya kapasitas *batch* memberikan ruang yang lebih besar untuk melakukan kombinasi beberapa pesanan terutama dalam kasus dengan *volume* kecil.

Pada kasus dengan kapasitas *batch* 150-200 secara umum memiliki nilai rata-rata *error* yang lebih besar dibandingkan dengan contoh kasus dengan kapasitas 100-150. Hal ini disebabkan karena adanya tambahan kapasitas ruang yang dapat diisi dalam satu *batch* akan tetapi menggunakan *range volume* yang sama yaitu 30-50, 30-100 dan 30-150. Sehingga, terjadi pertambahan *space* yang dapat diisi sebesar ± 50 dari batas maksimal *volume* 150 dan batas atas kapasitas *batch* 200. Pada contoh kasus yang menggunakan *volume* sedang, semakin besar space yang tersisa dalam satu *batch* akan lebih sulit untuk dikombinasikan.

5.4.2 Hasil HSAIGA dengan Hasil IGA

Dari hasil uji coba perbandingan metode HSAIGA dan IGA menggunakan contoh kasus data berukuran kecil pada **Tabel 5.10**, dapat diketahui bahwa HSAIGA memperoleh nilai rata-rata total biaya lebih optimal dibandingkan dengan IGA. Hasil pengujian IGA menggunakan data berukuran besar dapat dilihat pada **Tabel 5.11** dan **Tabel 5.12**. Apabila diambil nilai rata-rata *error* dari kedua tabel tersebut berdasarkan simulasi uji coba maka akan diperoleh nilai rata-rata *error* seperti pada **Gambar 5.9** berikut.



Gambar 5.9 Rata-rata *Error* IGA Berdasarkan Simulasi Uji Coba

Apabila nilai *error* rata-rata pada **Gambar 5.8** dan **Gambar 5.9** dibandingkan maka dapat diketahui bahwa nilai *error* rata-rata yang dihasilkan menggunakan metode HSAIGA secara keseluruhan memiliki nilai rata-rata *error* yang lebih kecil. Perbedaan antara HSAIGA dan IGA adalah pada metode HSAIGA, terdapat penggunaan metode *simulated annealing* di dalam IGA yang berfungsi sebagai *local search*. *Local search* (pencarian lokal) ditujukan untuk melakukan eksploitasi untuk mendapatkan solusi lain yang lebih baik dari solusi sekarang pada beberapa kromosom yang dipilih secara acak. Metode *simulated annealing*

dijalankan setelah proses *mutasi* dan *crossover* dilakukan. Penerapan metode *simulated annealing* sebagai *local search* terbukti dapat meningkatkan performa IGA.

5.4.3 Hasil HSAIGA dengan Penelitian Terdahulu

Pada penelitian Wang dan Luo (2016), metode hibridisasi *genetic algorithm* dan *simulated annealing* dilakukan dengan langkah-langkah sebagai berikut :

1. Inisiasi Parameter : T_0, popSize, cr dan *mr*, S = 0, counter p = 0.
2. Hasilkan kromosom sejumlah pop size sebagai populasi awal pop (*0*)
3. Hitung nilai *fitness* setiap kromosom dan pilih kromosom terbaik $\sigma *$. Jadikan nilai *fitness* kromosom terbaik menjadi solusi sementara terbaik, $S = f(\sigma *)$.
4. Terapkan operator genetik (seleksi, *crossover* dan mutasi) pada populasi (*n*).
5. Pilih kromosom dari populasi (*n*) sesuai dengan fungsi probabilitas Boltzmann untuk membuat generasi baru (*n* +1).
6. Perbaiki nilai temperatur berdasarkan $T_{n+1} = T_n \exp(-\theta t)$. Jadikan *n* = *n*+1.
7. Hitung nilai *fitness* setiap kromosom pada populasi (*n*+1) dan pilih kromosom terbaik σ'. Jadikan $S' = f(\sigma')$.
8. Apabila $S' > S$, jadikan $S = S'$.
9. Apabila T_n mencapai T_{min} maka hentikan perulangan. Apabila tidak, maka ulangi langkah ke 4.

Dari langkah-langkah tersebut, diketahui bahwa setiap iterasi pada metode *simulated annealing* dihitung nilai *fitness* terbaiknya S' dan nilai tersebut akan dibandingkan dengan nilai terbaik global S. Apabila nilai S' lebih baik dari S maka nilai terbaik global diubah menjadi $S = S'$. Dari penjelasan ini dapat diketahui bahwa metode *simulated annealing* pada penelitian Wang dan Luo (2016) tidak digunakan sebagai *local search* sebagaimana yang diterapkan pada metode HSAIGA.

Pada penerapan hibridisasi *genetic algorithm* dan *simulated annealing* oleh Wang dan Luo (2016), kelebihan metode *genetic algorithm* dan *simulated annealing* tidak dimanfaatkan dengan baik. *Genetic Algorithm* merupakan metode pencarian solusi secara global yang memiliki cakupan pencarian solusi yang luas (kemampuan eksplorasi yang baik) sedangkan metode *simulated annealing* mempunyai kelebihan dalam pencarian solusi lokal dan menghindari lokal optimal yang cukup baik (kemampuan eksploitasi). Agoritme SA mempunyai mekanisme untuk memberikan kesempatan bagi calon solusi yang kurang baik untuk diterima sebagai solusi sementara dengan probabilitas tertentu dengan tujuan agar tidak terjebak pada lokal optimal. Sedangkan pada penelitian Wang dan Luo (2016)

didasarkan pada pencarian solusi menggunakan SA sebagai metode utama dan operator *genetic algorithm* yaitu *crossover* dan mutasi yang merupakan pencarian berbasis populasi diterapkan pada metode SA tersebut, jadi penerapan hibridisasi tersebut tidak terlalu memanfaatkan kelebihan dari metode GA sebagai metode pencarian berbasis populasi. Rata-rata nilai *error* pada penelitian Wang dan Luo (2016) dapat dilihat pada **Tabel 5.13** dan **Tabel 5.14**.

Tabel 5.13 Rata-rata E*rror* Penelitian Wang dan Lou Kapasitas 100-150

$c \in [100, 150]$						
Size (*n*)	$v_i \in [30, 50]$		$v_i \in [30, 100]$		$v_i \in [30, 150]$	
	A.ER(%)	A.CpuT(s)	A.ER(%)	A.CpuT(s)	A.ER(%)	A.CpuT(s)
30	9.89	1.49	2.91	2.12	2.40	2.14
50	10.60	2.50	3.85	2.50	2.56	3.52
70	10.83	4.71	5.44	5.28	3.47	4.98
100	13.73	7.88	6.89	8.37	4.22	8.46
150	**15.10**	14.00	9.73	14.56	5.90	12.85

Sumber: (Wang & Luo, 2016)

Tabel 5.14 Rata-rata E*rror* Penelitian Wang dan Lou Kapasitas 150-200

$c \in [150, 200]$						
Size (*n*)	$v_i \in [30, 50]$		$v_i \in [30, 100]$		$v_i \in [30, 150]$	
	A.ER(%)	A.CpuT(s)	A.ER(%)	A.CpuT(s)	A.ER(%)	A.CpuT(s)
30	7.61	1.52	6.44	3.01	3.69	1.85
50	8.72	2.33	7.55	2.65	6.78	2.54
70	8.91	3.61	7.67	3.94	6.86	4.41
100	8.66	6.04	8.24	6.55	8.91	7.92
150	9.20	9.96	9.99	11.61	9.90	13.28

Sumber : (Wang & Luo, 2016)

Hasil pengujian yang telah dilakukan menggunakan metode HSAIGA pada **Table 5.8** dan **Tabel 5.9** menunjukkan bahwa nilai rata-rata *error* yang diperoleh lebih kecil dari 11% dengan waktu komputasi yang lebih cepat, sedangkan hasil pengujian pada penelitian Wang dan Lou (2016) secara keseluruhan memiliki nilai rata-rata *error* tidak lebih dari 16%. Dari kedua hasil tersebut dapat disimpulkan bahwa penggunaan *simulated annealing* sebagai *local search* pada metode HSAIGA dapat mengurangi nilai rata-rata *error* dan waktu komputasi pada penjadwalan produksi-distribusi dengan model *make to order*.

BAB 6 PENUTUP

6.1 Kesimpulan

Berdasarkan penelitian yang dilakukan terkait penjadwalan produksi-distribusi dengan model *make to order* menggunakan hibridisasi metode *simulated annealing* dan *improved genetic algorithm* (HSAIGA), maka dapat ditarik kesimpulan sebagai berikut:

1. Setelah mengetahui parameter-parameter apa saja yang digunakan dalam penjadwalan produksi-distribusi dengan model *make to order* maka permasalahan tersebut diformulasikan ke dalam sebuah model *mixed integer programming* sebagai berikut:

$$min\, Z = (\lambda_c + \eta_c).u + \beta_i.\left\{\sum_{i=1}^{n}(d_i - C_i)\right\}$$

persamaan tersebut digunakan untuk mencari biaya minimal produksi-distribusi dan memiliki 4 persamaan tambahan sebagai batasan.

2. Perancangan hibridisasi metode *simulated annealing* dan *improved genetic algorithm* dilakukan dengan cara mengabungkan metode *simulated annealing* kedalam *improved genetic algorithm* (IGA) sebagai *local search* untuk meningkatkan performa IGA. Perancangan hibridisasi ini menghasilkan algoritme dengan langkah-langkah sebagai berikut: Inisiasi populasi, perhitungan nilai *fitness*, reproduksi (*crossover* dan mutasi), local search (*simulated annealing*), dan eliminasi.
3. Hasil penerapan metode HSAIGA untuk menyelesaikan permasalahan produksi-distribusi yang dibandingkan dengan *lower bound* menghasilkan nilai rata-rata akurasi 89% atau rata-rata *error* yang tidak lebih dari 11%. Hasil pengujian ini lebih baik apabila dibandingkan dengan penelitan sebelumnya yang memiliki nilai rata-rata *error* tidak lebih dari 16%. Penurunan nilai rata-rata *error* membuktikan bahwa dengan menerapkan metode *simulated annealing* sebagai *local search* dapat mengoptimalkan solusi untuk permasalahan penjadwalan produksi-distribusi dengan model *make to order*.
4. Sebagai pembanding, peneliti melakukan percobaan menggunakan metode *improved genetic algorithm* untuk menyelesaikan permasalahan penjadwalan produksi-distribusi yang menghasilkan nilai rata-rata akurasi 87% atau rata-rata *error* tidak lebih dari 13%. Hal ini menunjukkan bahwa penerapan metode *simulated annealing* sebagai *local search* kedalam metode IGA dapat meningkatkan rata-rata nilai akurasi.

6.2 Saran

Saran yang dapat diberikan untuk pengembangan penelitian ini kedepannya adalah sebagai berikut:

1. Penelitian selanjutnya dapat berfokus pada objektif yang berbeda seperti meminimalkan *earliness penalty* pada contoh kasus data berukuran besar.
2. Dalam contoh kasus penjadwalan produksi-distribusi yang sama, metode *improved genetic algorithm* yang dipakai dalam penelitian ini dapat dikembangkan lagi untuk memperoleh hasil yang lebih optimal.