BAB II

LANDASAN TEORI

2.1 Pengertian Gudang

Gudang merupakan salah satu ruang yang dimanfaatkan dalam pabrik karena gudang adalah tempat penyimpanan baik itu barang yang akan dipergunakan dalam produksi, barang yang sudah diproduksi sampai barang tersebut dimintai sesuai dengan jadual produksi. Macam-macam gudang dapat dibedakan menurut karakteristik material yang disimpan yaitu sebagai berikut[WIN96]:

a. *Raw Material Storage*

   Gudang ini akan menyimpan setiap material yang dibutuhkan/digunakan untuk proses produksi. Lokasi dari gudang ini umumnya berada di dalam bangunan pabrik (*indoor*), akan tetapi untuk beberapa jenis bahan tertentu bisa juga diletakkan diluar bangunan pabrik (*outdoor*). Gudang ini disebut pula sebagai *stock room* karena fungsinya memang penyimpan stock untuk kebutuhan tertentu.

b. *Work In Proces Storage*

   Work In Proces Storage ini terdiri dari dua macam yaitu:

   - *Small amount materials*, yang akan diletakkan diantara stasiun kerja atau mesin atau pula suatu tempat yang berdekatan dengan lokasi operasi selanjutnya.

- *Large amount materials*, atau bahan-bahan yang akan disimpan dalam jumlah yang besar dan waktu yang relatif cukup lama yang mana disini lokasinya akan terletak terletak didalam *production area* yang ada.

c. *Finished Good Product Storage*

Finished Good Product Storage disebut juga *warehouse* yang fungsinya adalah penyimpanan produk-produk yang telah selesai dikerjakan.

Adapun tugas dan tanggungjawabnya adalah sebagai berikut:

- Penerimaan produksi jadi yang telah selesai diproduksi
- Penyimpanan produk jadi dengan sebaik-baiknya dan selalu siap pada saat permintaan masuk.
- Pengepakan (*packaging*) dari produk untuk dapat dikirim dengan aman.
- Menyelenggarakan administrasi pergudangan terutama untuk produk jadi.

Selain ketiga macam gudang tersebut diatas maka ada pula beberapa macam gudang lainnya yang perlu diketahui yaitu :

a. *Storage for Supplier*, yaitu untuk menyimpan *non-productive items* dan digunakan untuk menunjang fungsi dan kelancaran produksi seperti *packing materials*, *maintenance*, *supplier* dan *parts*, *office supplier* dan lain-lain.

b. *Finished Parts Storage*, yaitu gudang untuk menympan *parts* yang siap untuk dirakit. Gudang ini biasanya diletakkan berdekatan dengan

assembly area atau bisa juga ditempatkan secara terpisah di dalam *work in proces storage*.

c. *Salvage*, dalam sebagian besar proses produksi ada kemungkinan beberapa benda kerja akan salah dikerjakan (*mis-processed*) yang mana untuk ini memerlukan pengerjaan kembali untuk membetulkannya sehingga kualitas produksi tersebut diperbaiki. Untuk itu diperlukan suatu area guna menyimpan benda kerja ini sebelum proses kembali. Benda kerja yang tidak bisa diperbaiki akan menjadi skrap atau buangan limbah yang mana untuk ini harus diletakkan dalam lokasi tersendiri.

d. *Scrap dan Waste*, skrap adalah material atau komponen yang salah dikerjakan dan tidak bisa diperbaiki lagi. Material yang berupa skrap atau buangan ini biasanya akan dikumpulkan dan diletakkan dalam area yang terpisah dari pabrik dengan harapan akan bisa dijual ke pihak lain yang membutuhkan.

Dalam perencanaan luas area ataupun ruang yang diperlukan untuk gudang ini maka banyak pertimbangan yang harus dilakukan. Demikian juga banyak data dan informasi mengenai material yang akan disimpan (jumlah,berat,frekwensi pemakaian dan lain-lain) harus dikumpulkan dan dianalisa sehingga pada akhirnya dapat dicapai:

a. Pemakaian ruangan secara maksimal.

b. Pemakaian secara efektif dari waktu, tenaga kerja, dan peralatan yang dipergunakan.

c. Tersedianya setiap saat material/items untuk proses produksi.

d. Kemudahan proses pengambilan material terhadap bahaya perusakan seperti jamur, korosi dan lain-lain.

e. Penyimpanan rapi, teratur dan tata tertib dari gudang yang direncanakan.

Perencanaan luas area gudang pada penelitian ini akan ditinjau dalam dua dimensi, dengan demikian tidak sekedar mempertimbangkan luas area laintainya saja. Untuk menyimpan didalam gudang ini, ada beberapa metode yang umum dipakai yaitu menggunkan peti atau bak, papan lembaran (*shelves*).

2.2 Konsep Penyusunan Barang ke Dalam gudang

2.2.1 Metode Yang Berkembang

Pada pembahasan masalah penempatan barang persegi panjang pada sebuah persegi panjang yang lebih besar untuk meminimalisasi ketinggian nest. Semuanya harus dikemas dalam satu lembar objek.

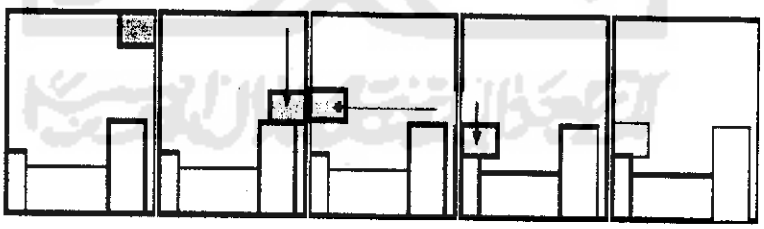
Dua dimensi yang berbeda dari masalah stock-cutting adalah NP keras dalam kombinasi ledakan pertempuran sebagai peningkatan permasalahan [GAR79]. Area ini telah menjadi sejak beberapa formulasi penelitian pada tahun 1950-an, dimana [PAU56] mengirimkan cetakan layout permasalahan. Dalam sifat yang lebih luas dari lokasi penelitian ini dan bermacam-macam permasalahannya, banyak peneliti memiliki pandanaan luar dan kategori bibliografi [DOW92], [SWE92], [DYK90], [COF84], [GOL76], [GIL66]. Pendekatan bisa lebih luas dengan tiga kategori: eksak, heuristic, dan metaheuristik.

Metode eksak ditemukan oleh [GIL61] yang diputuskan sebagai penelitian aplikasi industri nyata pertama dalam stock-cutting. Menggunakan teknik pemrograman linier untuk mendapatkan hasil optimal. Bagaimanapun, hanya masalah kecil yang dapat diselesaikan dalam waktu yang telah diperhitungkan. [CHR77] menggunakan metode pencarian pohon (tree) untuk menyelesaikan masalah stock-cutting guilotin dua-dimensi agar optimal, seperti juga [BEA85] dengan varian non guilotin. Bagaimanapun, sekali lagi, dengan contoh yang kompleks maka metode yang digunakan juga menjadi infeasible. [BEA85] membandingkan algoritma optimal dan heuristik menggunakan program dinamis. [HIF97] mengenalkan algoritma eksak yang dikembangkan dengan pendekatan yang digunakan oleh Christofides dan Whitlock. Baru saja, [CHU00] mengembangkan sebuah versi baru dari algoritma yang dikemukakan oleh [HIF97] yang menggunakan cabang terbaik dan batas pendekatan unruk memecahakan beberapa varian secara tepat dari masalah stock-cutting dua dimensi. Kelemahan terbesar dari metode ini adalah bahwa metode ini tidak bisa memberikan hasil yang baik dalam contoh masalah yang kompleks.. metode heuristik dibutuhkan untuk memberikan hasil yang baik, meskipun, yang pasti, tidak sepenting penyelesaian optimal.

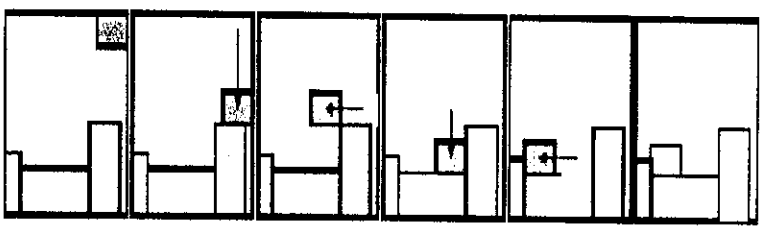
2.2.2 Pola Penyusunan dengan Metode Heuristik

Ada banyak pendekatan heuristik yang diformulasikan untuk memproduksi pengemasan yang bagus dengan frame waktu yang pantas, sebesar masalah stock-cutting. [ALB79] menggunakan sebuah metode heuristik yang memuat persegi

yang serupa kedalam sebuah bidang masalah yang besar dengan contoh range antara 400-4000 persegi.. [BEN82] membuat sebuah solusi heuristik yang mencapai garis kerugian antara 2% dan 5% dengan menyortir persegi dan kemudian menyusun mereka ke dalam tumpukan. Kebanyakan dokumen tentang pendekatan heuristik adalah metode Bottom-Left (BL) dan Bottom-Left-Fill (BLF) [BAK80], [CHA83]. [JAK96] menggunakan metode Bottom-Left yang mengambil input sebuah data persegi panjang dan meletakkannya satu persatu kedalam lembaran stok. Strategi penempatan,pertama persegi diletakkan pada lokasi atas kanan dan berturut-turut menggesernya kebawah sampai ke kiri jika memungkinkan (Gambar 2.1). [LUI96] gambar 2.2 mengembangkan perbaikan Bottom-Left heuristik, memprioritaskan perpindahan bagian bawah sehingga bentuknya hanya bergeser kearah kiri dan tidak ada perpindahan kebawah. Seperti yang terlihat , tidak seperti metode Jakob, yang selalu menjadikan persegi terakhir sebagai patokan solusi optimal.

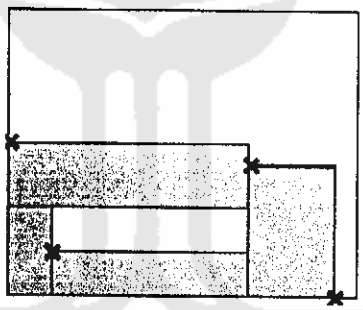


Gambar 2.1 Metode bottom-left [JAC96]



Gambar 2.2 Perbaikan metode bottom-left [LIU99]

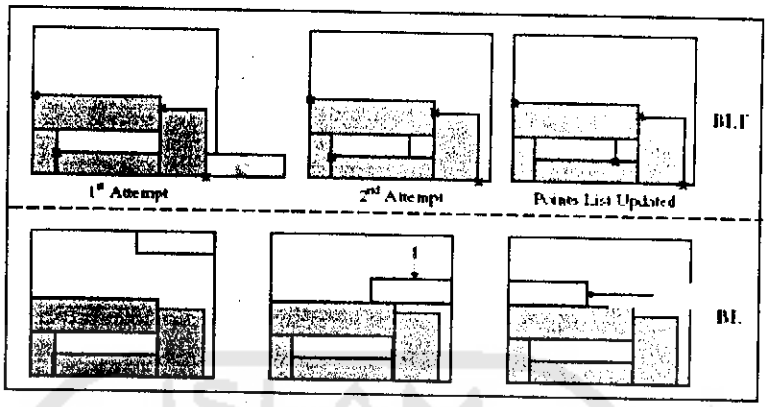
Metode kedua, Bottom-Left-Fill, merupakan modifikasi dari metode Bottom-Left. Implementasi pertama mempertahankan data lokasi untuk mengindikasikan dimana bentuk-bentuk akan diletakkan. Saat meletakkan bentuk-bentuk, algoritma dimulai dengan poin paling rendah dan paling kiri,tempat bentuk dan justifies kiri, kemudian dicek apakah bentuk tersebut akan bertindihan dengan bentuk yang lain dan membatasi lembaran. Jika tidak bertindihan,bentuk tersebut diletakkan dan poin di *update* untuk mengindikasikan poin yang baru. Jika bentuk bertindihan, poin selanjutnya dalam daftar poin adalah memilih bentuk sampai tidak terjadi saling tindih. Gambar 2.3 menunjukkan contoh dari poin penempatan.



Gambar 2.3 Lokasi tempat penyimpanan untuk satu implementasi dari bottom-left fill

Dalam gambar 2.3, ketika meletakkan bentuk kelima algoritma harus diletakkan di poin pertama paling bawah (dengan poin yang sama tinggi dengan yang telah diselesaikan pertama di kiri). Oleh karena itu, Bottom-Left-Fill bisa mengatasi masalah lubang oleh poin lokasi yang memungkinkan. Kita dapat menunjukkan perbedaan antara dua penempatan heuristik dengan melakukan strategi ketika menambahkan bentuk kelima ke atas pengemasan (gambar 2.4).

Gambar 2.4 menunjukkan bahwa Bottom-Left-Fill bisa untuk mengisi lobang menggunakan *fitting* persegi panjang dalam pengemasan, dimana menggunakan Bottom-Left menghasilkan lubang pada akhir pengemasan. Lubang ini tak pernah diisi (kecuali setelah persegi di *fitting*), oleh karena itu, space akan bisa diputuskan untuk dibuang. Aspek terpenting dalam algoritma ini adalah urutan penyediaan persegi panjang dapat sangat mempengaruhi kualitas solusi. Dalam eksperimen antar algoritma, [HOP01] menemukan bahwa Bottom-Left-Fill memperbaiki Bottom-Left lebih dari 25% dan sebelum membentuk bentukan ini dengan mengurangi lebar atau mengurangi tinggi untuk memperbesar kualitas pengemasan algoritma lebih dari 10% dibandingkan secara random. Keuntungan utama menggunakan Bottom-Left adalah kompleksitas waktu dari $O(N^2)$ [HOP01]. Kelemahan algoritma Bottom-Left-Fill adalah buruknya kompleksitas waktu dari $O(N^3)$ [CHA83]. Dengan konsekuensi, eksekusi waktu dapat menjadi lebih besar untuk Bottom-Left-Fill dengan peningkatan besar masalah. Mengesampingkan keistimewaan dari pendekatan ini bahwa pengemasan masing-masing bentuk tergantung dari aturan set. Oleh karena itu, solusi bisa jadi sama jika penyediaan dengan persegi yang identik secara berurutan. Mencoba untuk memproduksi pengemasan yang berkualitas baik dalam satu pekerjaan, mengurangi waktu yang diperlukan untuk mencapai solusi.



Gambar 2.4 Perbandingan antara metode BL dan BLF dengan menambahkan persegi panjang

2.3 Algoritma Genetika

2.3.1 Pengertian Algoritma Genetika

Algoritma genetika adalah sebuah metode algoritma yang bertujuan untuk mencari solusi suatu masalah dengan pola yang paling optimal. Metode ini pertama kali dikembangkan oleh John Holland beserta koleganya dan murid-muridnya di Universitas Michigan.

Selanjutnya algoritma genetika banyak dikembangkan dan digunakan dalam ilmu kedokteran dan biologi, yaitu untuk menemukan susunan-susunan gen yang terbaik dalam tubuh manusia, binatang maupun tumbuhan dengan menggabungkan antara metode *reproduction* (reproduksi), *CrossOver* (pindah silang), mutation (mutasi) dan termination (terminasi).

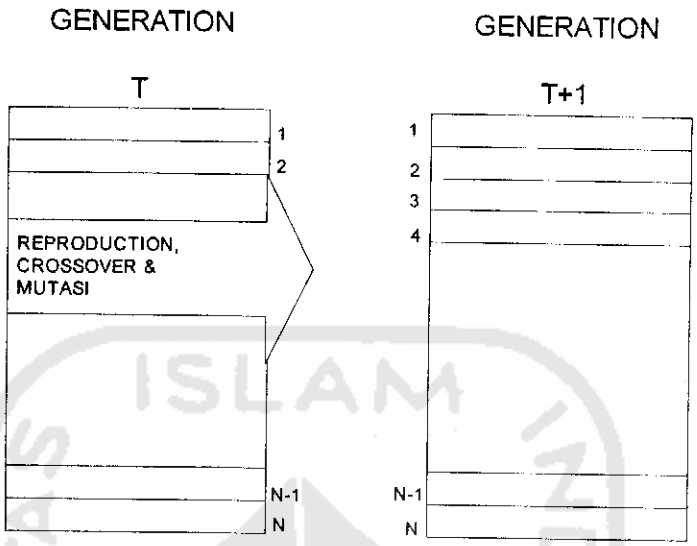
Solusi yang didapat dari berbagai penggabungan metode tersebut dinamakan *chromosome* (kromosom) dengan *fitness* (nilai kemampuan untuk bertahan hidup pada generasi selanjutnya) tertentu, dimana setiap kromosom yang

terdiri dari susunan banyak *gene* (gen) mewakili satu individu dalam setiap *population* (populasi) pada suatu *generation* (generasi). Pencarian susunan gen-gen terbaik ini membutuhkan waktu yang cukup lama seiring dengan proses evolusi yang ada.

Algoritma genetika memiliki perbedaan mendasar dalam empat hal jika dibandingkan dengan metode algoritma pencarian solusi lainnya yaitu:

- Algoritma genetika memanipulasi himpunan parameter terkode, bukan secara langsung pada parameter tersebut.
- Algoritma genetika melakukan pencarian dari suatu populasi titik, bukan dari satu titik tunggal yang bersifat heuristik.
- Algoritma genetika memanfaatkan informasi yang ada secara langsung dan tidak memanfaatkan derivasi atau pengetahuan tambahan yang lainnya.

Kebanyakan penelitian algoritma genetika difokuskan pada tiga operator utamanya yaitu reproduksi, pindah silang dan mutasi, disamping operator terminasi yang berfungsi untuk menentukan kapan proses iterasi harus berakhir. Berikut ini adalah struktur umum dari algoritma genetika:



Gambar 2.5 Struktur umum dari algoritma genetika

Pada gambar diatas dapat dijelaskan bahwa individu-individu *parent* dalam "GENERATION T" akan mengalami proses-proses algoritma genetika yaitu: reproduction, crossover, dan mutasi untuk menghasilkan individu-individu baru (*offspring*) pada "GENERATION T+1".

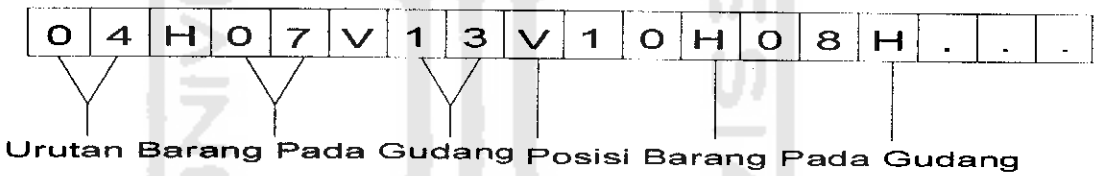
2.3.2 Pembuatan Populasi Awal

Populasi merupakan kumpulan beberapa individu. Semua populasi dalam GA ini asalnya dari 1 populasi yaitu populasi awal. Solusi atau kromosom terbaik dari populasi awal ini akan dipertahankan, dan akan mengalami proses evolusi untuk mendapatkan kemungkinan solusi yang lebih baik.

Pembuatan populasi awal ini dilakukan melalui proses pemilihan secara acak dari seluruh solusi yang ada. Pemilihan acak ini menyebabkan populasi awal dari algoritma genetika tidak akan sama dalam setiap percobaan, meskipun semua nilai variabel yang digunakan sama.

Pada kasus pengisian barang kegudang yang menjadi individu adalah urutan barang yang akan dimasukkan ke dalam gudang. Selain urutan barang, di dalam setiap individu juga tersimpan informasi bagaimana posisi barang ketika masuk kedalam gudang (Vertikal atau Horisontal). Satu solusi akan berisi urutan dari seluruh barang yang ada. Karena itu nantinya sebelum menjalankan algoritma genetika pada perangkat lunak ini, harus terlebih dahulu mengisikan barang-barang yang ada beserta dengan spesifikasinya (panjang, lebar).

Dalam hal ini populasi adalah sebuah string yang berisi gen yang berjumlah sesuai dengan barang pada tiap kategori. Karena jumlah panjang gen 3 karekter, maka panjang string adalah 3 x n, dengan n merupakan jumlah barang yang akan dimasukkan ke dalam gudang.



Gambar 2.6 Visualisasi Individu dalam Populasi

2.3.3 Perhitungan Nilai *Fitness*

Untuk mengetahui baik tidaknya solusi yang ada pada suatu individu, setiap individu pada populasi harus memiliki nilai pembandingnya (*Fitness Cost*). Melalui nilai pembanding inilah akan didapatkan solusi terbaik dengan cara pengurutan nilai pembanding dari individu-individu dalam populasi. Solusi terbaik ini akan dipertahankan, sementara solusi lain diubah-ubah untuk

mendapatkan solusi yang lain, melalui tahap CrossOver dan mutasi (mutation). Berikut ini adalah rumus Nilai fitness [LIM04]:

$$\text{NILAI FITNESS} = \frac{\text{Luas Barang}}{\text{Luas Okupansi}} * 100\% \quad \text{.....(2.1)}$$

Luas okupansi adalah luas segiempat luar yang ada pada gudang, luasan ini adalah area pada gudang yang dianggap terpakai dalam penyimpanan barang. Seperti terlihat pada gambar 2.7 dibawah ini.

Gambar 2.7 Kondisi Gudang untuk mencari nilai fitness

2.3.4 Pemilihan (Seleksi) Induk

Operasi seleksi digunakan untuk memilih kromosom dalam proses *crossover* nantinya. Jenis operasi seleksi yang biasa digunakan antara lain seleksi secara acak, seleksi roda *roulette,* selcksi ranking, dan seleksi turnamen. Masing-masing seleksi menerapkan penekanan selektif, yaitu memilih kromosom berdasarkan nilai *fitness*nya, kecuali pada seleksi acak. Seleksi berguna untuk mengeksploitasi ruang pencarian kandidat solusi. Seleksi disini adalah pemilihan-pemilihan individu dengan nilai *fitness* terbaik untuk dijadikan induk dalam menghasilkan individu-individu baru.

Pemilihan individu-individu terbaik itu dapat dilakukan dengan menggunakan mekanisme mesin *roulette* dimana semua individu ditempatkan dalam suatu lingkaran. Prosentase luas dari setiap individu pada lingkaran *roulette* didasarkan pada nilai *fitness*nya. Mengingat fungsi tujuan adalah memaksimalkan nilai *fitness*, dengan nilai *fitness* individu makin besar maka makin besar luas daerah yang diperoleh dalam lingkaran *roulette* sehingga makin besar pula kemungkinan individu tersebut terpilih. Kemudian mesin *roulette* diputar dimana jarumnya kemudian akan menunjuk individu yang terpilih. *Crossover* berguna untuk mengeksploitasi ruang pencarian kandidat solusi.

2.3.5 CrossOver (Perkawinan Silang)

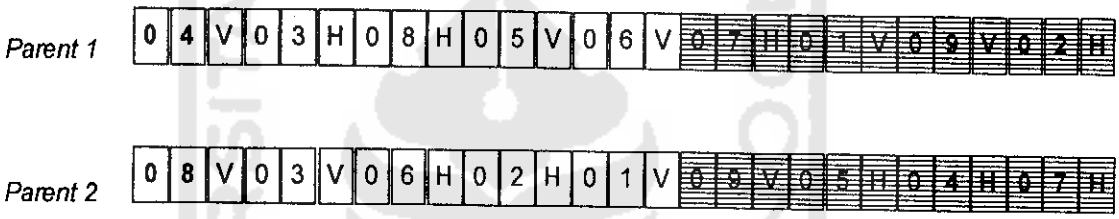
Setelah dilakukan pemilihan induk, maka antar kromosom induk yang terpilih dilakukan persilangan untuk mendapatkan kromosom – kromosom anak yang berbeda dengan induknya, yang juga merepresentasikan alternatif solusi dari permasalahan yang dihadapi.

Pada operasi *CrossOver* dibutuhkan dua induk untuk menghasilkan keturunan. Untuk itu setiap *CrossOver* diperlukan dua induk dari hasil seleksi, jika hasil seleksi merupakan bilangan ganjil maka dibuang satu kromosomnya sehingga menjadi bilangan genap. Jenis *CrossOver* antara lain *Partial Mapped CrossOver* (PMX), *Order CrossOver* (OX), *Position Based CrossOver* (PBX), *Order Based CrossOver* (OBX).Dengan masing-masing penjelasan sebagai berikut [GEN97]:

1. *Partial Mapped CrossOver*

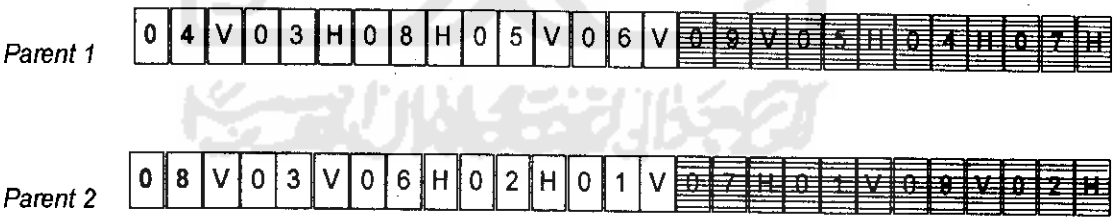
Partial Mapped CrossOver (PMX) diperkenalkan oleh Goldberg dan Lingle. PMX menggunakan prosedur khusus untuk mengatasi terjadinya *offspring* yang ilegal dari persilangan dua titik. Inti dari PMX adalah persilangan dua titik ditambah dengan prosedur perbaikan. Cara kerjanya sebagai berikut :

a. Pilih dua posisi sepanjang *string* secara acak bebas. *Substring* yang didapat dari dua posisi yang telah dipilih disebut *mapping sections.*



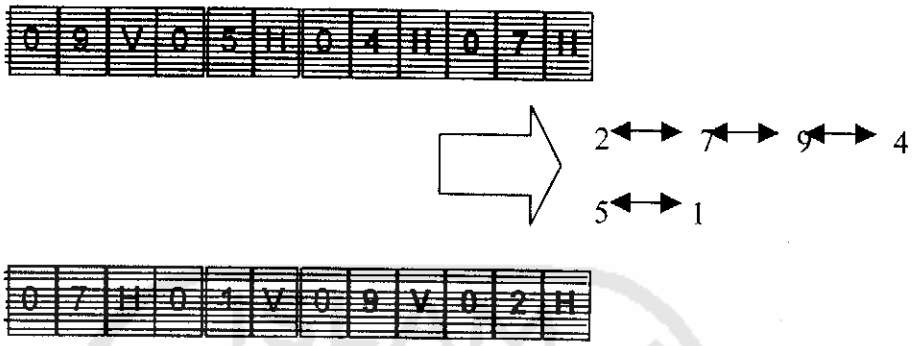
Gambar 2.8a Posisi awal kromosom sebelum operasi *CrossOver* dengan *PMX*

b. Tukarkan posisi dan *substring* antara kedua *parent* untuk menghasilkan *protochildren.*



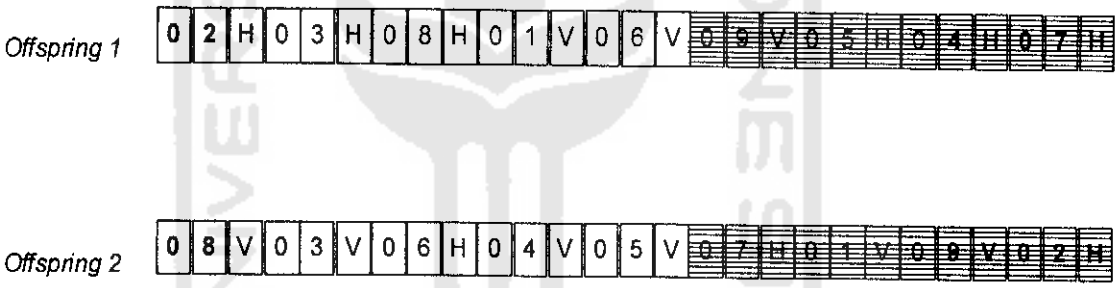
Gambar 2.8b Proses penyilangan *substring* antara *parent* 1 dan 2

c. Analisis *mapping relationship* diantara dua bagian *mapping* tersebut.



Gambar 2.8c Penentuan relasi *mapping*

d. Buat *offspring* menjadi legal jika dari langkah 2 ditemukan *offspring* yang tidak memenuhi kriteria legal (*illegatiamate).*



Gambar 2.8d *Legal offspring*

2. *Order CrossOver*

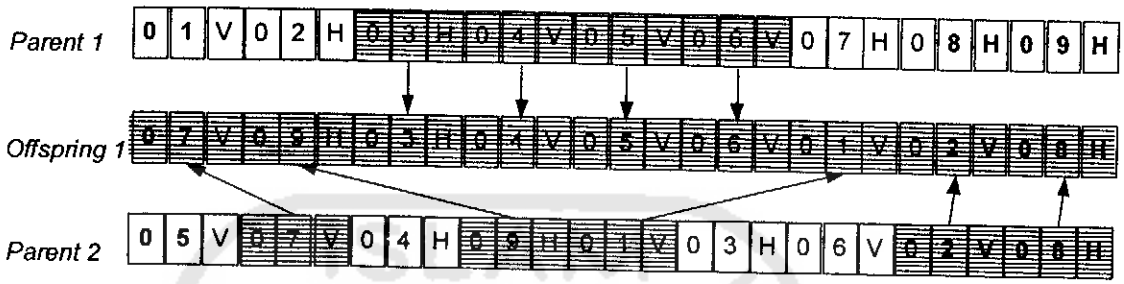
Order CrossOver (OX) dikenalkan oleh Davis. Cara kerjanya sebagai berikut :

a. Pilih *substring* secara bebas dari *parent.*

b. Copy *substring* ke *offspring* yang akan digenerasi dengan posisi yang sama.

c. Abaikan gen dengan nilai yang sama dengan yang sudah ada di langkah 2. Hasilkan urutan yang memiliki *substring* yang dibutuhkan oleh *protochild.*

d. Tempatkan sisa *substring parent* lainnya dengan posisi dari kiri ke kanan menurut aturan dalam menggenerasi *offspring.*



Gambar 2.9 *CrossOver* dengan *Order CrossOver*

3. *Position Based CrossOver*

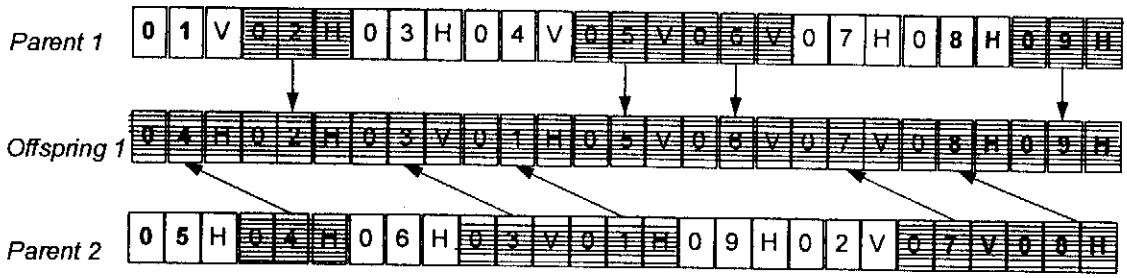
Position Based CrossOver (PBX) dikenalkan oleh Syswerda, pada intinya sejenis representasi *CrossOver* pada umumnya, dengan prosedur pembenahan. *Position Based* disebut juga variasi dari OX dengan pemilihan *substring* secara tidak urut. Cara kerjanya sebagai berikut :

a. Pilih satu set posisi *parent* secara acak.

b. Copy *substring* sesuai dengan posisi awalnya ke *protochild* sesuai dengan posisi awal *parent* asalnya.

c. Abaikan *offspring* dengan nilai yang sama dengan yang sudah ada pada langkah 2. Hasilkan urutan yang memiliki *substring* yang dibutuhkan oleh *protochild.*

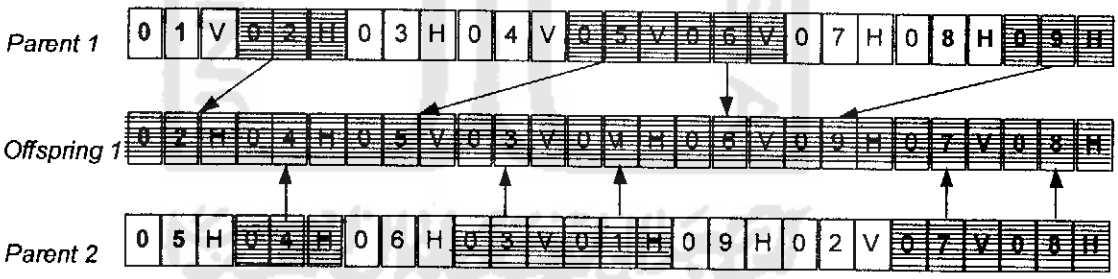
d. Tempatkan *substring* dengan posisi acak ke *protochild* dengan posisi dari kiri ke kanan menurut aturan dalam menggenerasi *offspring.*



Gambar 2.10 *CrossOver* dengan *Position Based CrossOver*

4. *Order Based CrossOver*

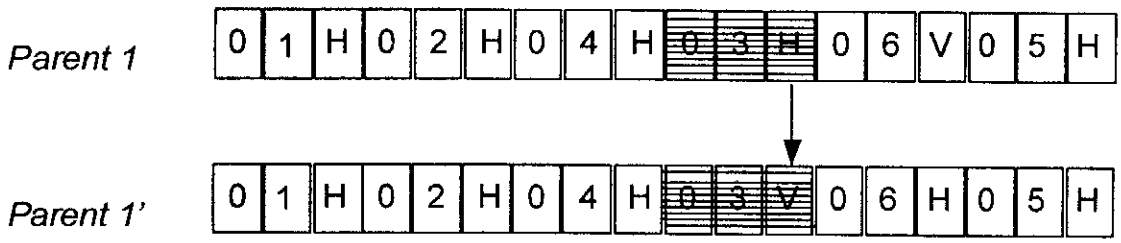
Order Based CrossOver (OBX) dikenalkan oleh Syswerda, merupakan variasi dari *Position Based CrossOver*, perbedaannya terletak pada penempatan *substring* dari *parent* yang pertama yang ditempatkan sesuai order atau *substring* yang belum ada di *protochild* dari posisi yang telah ditempati *substring* dari *parent* kedua.



Gambar 2.11 *CrossOver* dengan *Order Based CrossOver*

2.3.6 Mutasi (Mutation)

Operasi mutasi melakukan perubahan nilai gen – gen pada kromosom dengan cara menyisipkan kode informasi ke dalam suatu individu. Mutasi bertujuan untuk menjamin bahwa algoritma genetika melakukan eksplorasi pada

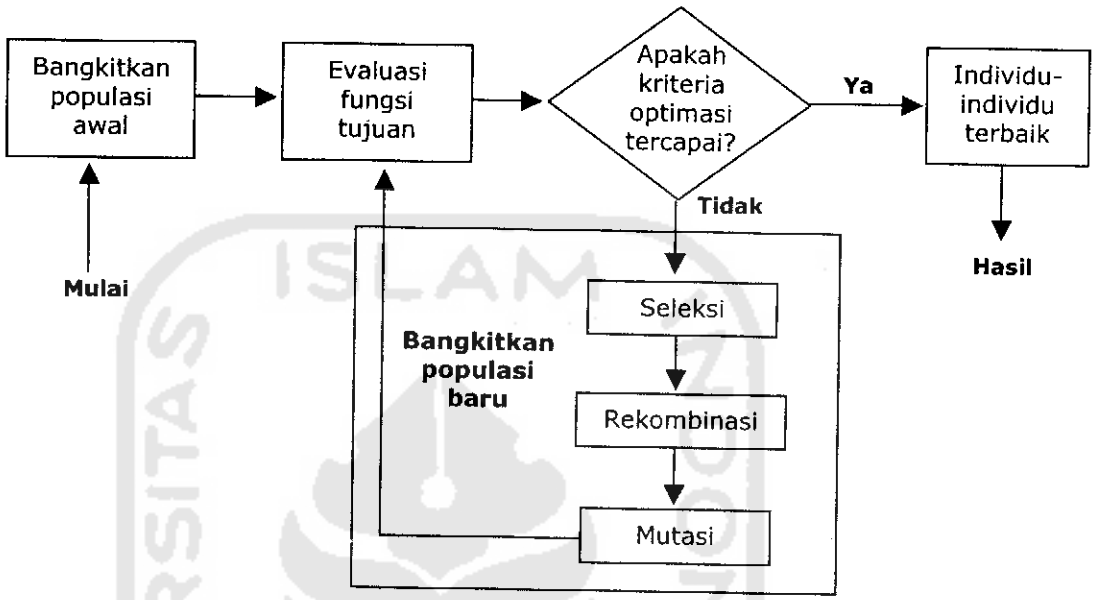


Gambar 2.13 Operasi mutasi dengan *Rotasi Posisi*

2.3.7 Pembaharuan Generasi

Proses terakhir setelah operasi mutasi adalah pembaharuan generasi, yang dilakukan untuk mengganti populasi lama dengan populasi baru dengan harapan bahwa populasi baru tersebut akan mempunyai nilai *fitness* yang lebih baik daripada populasi lama. Pembaharuan tersebut dapat dilakukan dengan menggantikan secara keseluruhan populasi lama dengan populasi baru yang disebut *generational update*, atau menggantikan sebagian kromosom dalam populasi lama dengan kromosom – kromosom baru yang disebut *continous update*. Jika pendekatan selektif yang dilakukan cenderung mempertahankan kromosom yang baik dari generasi lama ke generasi baru dan hanya menggantikan kromosom yang nilai *fitness* nya kurang baik, maka jenis pembaharuan ini disebut *steady state update*.

Secara umum, diagram alir algoritma genetika sederhana seperti terlihat pada gambar 2.14



Gambar 2.14 Diagram alir proses algoritma genetika