BAB 2

LANDASAN TEORI

2.1. Definisi Warna

Warna [5] merupakan elemen terpenting pada sebuah objek berupa citra dimana elemen warna sangat dipengaruhi oleh intensitas cahaya. Warna sendiri dapat dilihat oleh indera penglihatan manusia dikarenakan adanya pantulan cahaya pada suatu permukaan benda dan dipengaruhi oleh pigmen yang terdapat pada permukaan benda tersebut.

Pada sebuah citra digital, warna dapat dibagi menjadi beberapa model warna yang digambarkan pada sistem koordinat berdimensi tiga. Model warna yang umum dipakai adalah RGB (digunakan pada monitor), CMY (digunakan pada printer berwarna), YIQ (digunakan pada siaran televisi berwarna), HSV, CIEXYZ, CIE LUV, CIE Lab, dan Munsell. Model warna dibedakan berdasarkan sistem koordinat maupun gamut warna.

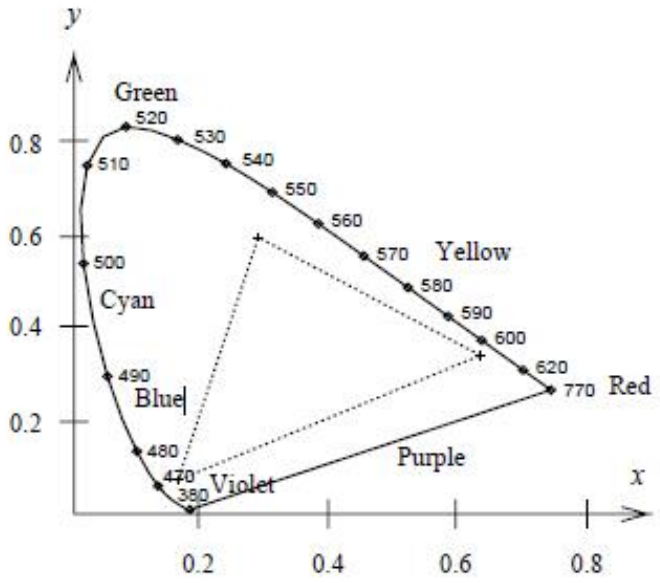
2.1.1. Model Warna RGB

Model warna RGB [5] digunakan oleh kebanyakan monitor berwarna. Monitor berwarna dapat menghasilkan 16 juta warna berbeda berdasarkan perpaduan warna dari ketiga *chanel* warna RGB yaitu merah, hijau dan biru. Gamut dari monitor berwarna RGB seringkali digambarkan sebagai unit kubus sehingga warna monitor sepenuhnya dikombinasikan oleh warna merah jenuh, kuning, hijau, cyan, biru, dan magenta. Pada sudut-sudut kubus terdapat warna hitam dan putih dimana secara diagonal menjadi degradasi warna putih menuju keabu-abuan hingga menjadi hitam. Sesuai dengan kromatisitas triad, gamut monitor dapat digambarkan sebagai segitiga kromatisitas tapal kuda yang dapat dilihat pada gambar 2.1. Sejumlah perpaduan warna [11] yang merupakan bagian dari 16 juta kemungkinan warna pada model warna RGB memiliki penamaan seperti yang dilampirkan pada lampiran A.1.

2.1.2. Model Warna CIE XYZ

Berbagai macam warna dapat dihasilkan melalui pencahayaan warna merah, hijau dan biru. Kurva pencocokan warna r(λ), g(λ), dan b(λ) menggambarkan angka dari RGB yang dibutuhkan untuk mencocokkan warna dengan panjang gelombang dominan λ. Hal terpenting adalah kurva merah r(λ) adalah tidak eksis pada 438 nm hingga 546 nm atau dikatakan negatif pada kurva tersebut, yang berarti bahwa warna pada daerah tersebut tidak dapat diproduksi melalui kombinasi positif dari R, G dan B.

Pada tahun 1931 Komisi Internationale de L' Eclairge (CIE) membahas masalah bobot negatif dalam model warna RGB. Komisi ini mendefinisikan tiga primari baru yang disebut sebagai X, Y dan Z [5] untuk menggantikan RGB. Pencocokan warna kurva x(λ), y(λ), dan z(λ) menggambarkan angka dari XYZ yang dibutuhkan untuk mencocokkan warna dengan panjang gelombang dominan λ. Kurva ini dirancang untuk menjadi kombinasi linear dari r(λ), g(λ), dan b(λ). Tidak satu pun dari x(λ), y(λ), ataupun z(λ) negatif dalam kisaran 380nm hingga 780nm, yang berarti setiap warna dapat diproduksi dengan kombinasi positif X, Y dan Z.



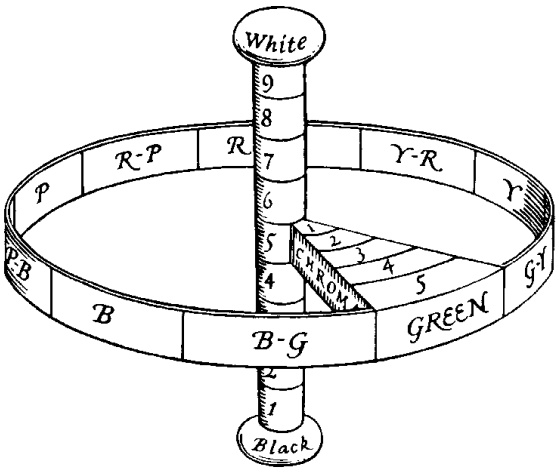
Gambar 2.1. Kromatisasi warna CIE [5]

2.1.3. Model Warna CIE LUV

Masalah yang timbul pada model warna CIE XYZ [5] adalah kurangnya keseimbangan perseptual warna. Warna yang memiliki jarak yang sama antara satu sama lain secara perseptual tidak selalu berjarak sama. Oleh karena alasan tersebut pada tahun 1967, CIE mengusulkan model warna CIE LUV. CIE LUV merupakan sebuah model warna yang memiliki perseptual warna dengan jarak yang sama.

2.1.4. Model Warna Munsell

Model warna Munsell [5] diusulkan pada tahun 1898 oleh Albert H. Munsell yang kemudian direvisi kembali pada tahun 1943 oleh Optical Society of America untuk lebih mendekati keseimbangan warna. Warna pada model warna Munsell ditentukan dengan menggunakan tiga dimensi elemen yaitu hue, kroma (*chroma*), dan nilai (*value*) seperti pada gambar 2.2.



Gambar 2.2. Model warna Munsell [7]

Hue mengacu pada identifikasi keunikan warna yang digambarkan oleh *band* melingkar yang dibagi menjadi sepuluh bagian. Munsell memberikan nama pada setiap bagian *band* tersebut sebagai merah, kuning-merah, kuning, hijau-kuning, hijau, biru-hijau, biru, ungu-biru, ungu dan merah-ungu (R, Y, GY, G,

BG, B, PB, P, dan RP). Setiap bagian tersebut dapat dibagi kembali menjadi sepuluh subbagian sesuai dengan kehalusan hue yang dibutuhkan.

Nilai (*value*) mengacu pada tingkat kecerahan atau kegelapan warna yang dapat dibagi menjadi sebelas bagian (penomoran dimulai dari 0 hingga 10). Untuk warna yang gelap memiliki nilai yang rendah sedangkan warna yang cerah memiliki nilai yang tinggi.

Kroma (*chroma*) mendefinisikan kekuatan atau kelemahan warna. Chroma diukur dalam langkah-langkah yang bernomor dimulai dari angka 1. Warna lemah memiliki nilai kroma yang rendah sedangkan warna yang kuat memiliki nilai kroma yang tinggi. Kroma bernilai nol adalah warna abu-abu sehingga nilai nol pada kroma dapat diabaikan.

2.2. Pengertian *Clustering*

Salah satu metode yang diterapkan dalam pendeteksian warna adalah *clustering*. *Clustering* [1] merupakan sebuah metode yang membagi data ke dalam grup-grup yang mempunyai obyek yang karakteristiknya sama. Garcia-Molina et al. menyatakan *clustering* adalah mengelompokkan item data ke dalam sejumlah kecil grup sedemikian sehingga masing-masing grup mempunyai sesuatu persamaan yang esensial.

Clustering adalah metode data mining yang *unsupervised* yang berarti tidak ada satu atributpun yang digunakan untuk memandu proses pembelajarannya sehingga seluruh atribut input diperlakukan sama

2.2.1. Kategori *Clustering*

Clustering [1] dapat dibagi kedalam dua kelompok, yaitu *hierarchical* dan *partitional clustering*. *Partitional Clustering* disebutkan sebagai pembagian obyek-obyek data ke dalam kelompok yang tidak saling overlap sehingga setiap data berada tepat di satu *cluster*. *Hierarchical clustering* adalah sekelompok *cluster* yang bersarang seperti sebuah pohon berjenjang (hirarki).

1. *Partitioning algorithms*: algoritma dalam kelompok ini membentuk bermacam partisi dan kemudian mengevaluasinya dengan berdasarkan beberapa kriteria.
2. *Hierarchy algorithms*: pembentukan dekomposisi hirarki dari sekumpulan data menggunakan beberapa kriteria.
3. *Density-based*: pembentukan *cluster* berdasarkan pada koneksi dan fungsi densitas.
4. *Grid-based*: pembentukan *cluster* berdasarkan pada struktur *multiple-level granularity*
5. *Model-based*: sebuah model dianggap sebagai hipotesa untuk masing-masing *cluster* dan model yang baik dipilih diantara model hipotesa tersebut.

2.3. Algoritma K-Means

K-means [1] merupakan salah satu metode *clustering* non hirarki yang berusaha mempartisi data yang ada ke dalam bentuk satu atau lebih *cluster*. Metode ini mempartisi data ke dalam *cluster* sehingga data yang memiliki karakteristik yang sama dikelompokkan ke dalam satu *cluster* yang sama dan data yang mempunyai karateristik yang berbeda di kelompokan ke dalam *cluster* yang lain. Secara umum algoritma dasar dari K-Means adalah sebagai berikut :

1. Tentukan jumlah *cluster*
2. Alokasikan data ke dalam *cluster* secara *random*
3. Hitung *centroid*/rata-rata dari data yang ada di masing-masing *cluster*
4. Alokasikan masing-masing data ke *centroid*/rata-rata terdekat
5. Kembali ke Step 3, apabila masih ada data yang berpindah *cluster* atau apabila perubahan nilai *centroid* ada yang di atas nilai *threshold* yang ditentukan.

Distance space digunakan untuk menghitung jarak antara data dan *centroid*. Adapun persamaan yang dapat digunakan salah satunya yaitu *Euclidean Distance Space*. *Euclidean distance space* sering digunakan dalam perhitungan jarak, hal ini dikarenakan hasil yang diperoleh merupakan jarak terpendek antara dua titik yang diperhitungkan. Adapun persamaannya adalah sebagai berikut :

$$d_{ij} = \sqrt{\sum_{k=1}^{p} \{x_{ik} - x_{jk}\}^2} \quad \text{..... (2.1)}$$

dimana,

d_{ij} = jarak *euclidean* antara data dan *centroid*

x_{ik} = data

x_{jk} = *centroid*/titik pusat *cluster*

p = dimensi data

k = jumlah *centroid*

2.4. Algoritma Genetika

Algoritma Genetika [4] adalah algoritma pencarian heuristik yang didasarkan atas mekanisme evolusi biologis. Keberagaman pada evolusi biologis adalah variasi dari kromosom antar individu organisme. Variasi kromosom ini akan mempengaruhi laju reproduksi dan tingkat kemampuan organisme untuk tetap hidup. pada dasarnya ada 4 kondisi yang sangat mempengaruhi proses evaluasi yaitu :

1. kemampuan organisme untuk melakukan reproduksi
2. keberadaan populasi organisme yang bisa melakukan reproduksi
3. keberagaman organisme dalam suatu populasi
4. perbedaan kemampuan untuk *survive*

Individu yang lebih kuat (fit) akan memiliki tingkat survival dan tingkat reproduksi yang lebih tinggi jika dibandingkan dengan individu yang kurang fit. Pada kurun waktu tertentu (sering dikenal dengan istilah generasi), populasi secara keseluruhan akan lebih banyak memuat organisme yang fit.

Algoritma Genetika pertama kali dikembangkan oleh John Holland dari universitas Michigan (1975). John Holland mengatakan bahwa setiap masalah yang berbentuk adaptasi (alami maupun buatan) dapat diformulasikan dalam terminologi genetika. Algoritma genetika adalah simulasi dari proses evolusi Darwin dan operasi genetika atas kromosom.

2.4.1. Stuktur Umum Algoritma Genetik

Pada algoritma ini [4] teknik pencarian dilakukan sekaligus atas sejumlah solusi yang mungkin yang dikenal dengan populasi. Individu yang terdapat dalam satu populasi disebut istilah kromosom. Kromosom ini merupakan suatu solusi yang masih berbentuk simbol. Populasi awal dibangun secara acak, sedangkan populasi berikutnya merupakan hasil evolusi kromosom-kromosom melalui iterasi yang disebut dengan istilah generasi. Pada setiap generasi, kromosom akan melalui proses evaluasi dengan menggunakan alat ukur yang disebut dengan fungsi *fitness*.

Nilai *fitness* dari suatu kromosom akan menunjukkan kualitas kromosom dalam populasi tersebut. Generasi berikut dikenal dengan istilah anak (*offspring*) terbentuk dari gabungan 2 kromosom generasi sekarang yang bertindak sebagai induk (parent) dengan menggunakan operator penyilangan (crossover). Selain operator penyilangan, suatu kromosom juga dapat dimodifikasi dengan menggunakan operator mutasi.

Populasi generasi yang baru dibentuk dengan cara menyeleksi nilai *fitness* dari kromosom induk (parent) dan nilai *fitness* dari kromosom anak (offspring), serta menolak kromosom-kromosom yang lainnya sehingga ukuran populasi (jumlah kromosom dalam suatu populasi) konstan. Setelah melalui beberapa generasi, maka algoritma ini akan konvergen ke kromosom terbaik.

2.4.2. Algoritma Genetika Komponen-komponen Utama

Ada 6 komponen utama dalam algoritma genetika [4], yaitu :

1. Teknik Penyandian

Teknik penyandian disini meliputi penyadian gen dari kromosom. Gen merupakan bagian dari kromosom. Satu gen biasanya akan mewakili satu variabel. Gen dapat dipresentasikan dalam bentuk: string bit, pohon, array bilangan real, daftar aturan, elemen permutasi, elemen program, atau representasi lainnya yang dapat diimplementasikan untuk operator genetika.

2. Prosedur Inisialisasi

Ukuran populasi tergantung pada masalah yang akan dipecahkan dan jenis operator genetika yang akan diimplementasikan. Setelah ukuran populasi ditentukan, kemudian harus inisialisai terhadap kromosom yang terdapat pada populasi tersebut. Inisialisasi kromosom dilakukan secara acak, namun demikian harus tetap memperhatikan domain solusi dan kendala permasalahan yang ada.

3. Fungsi Evaluasi

Ada 2 hal yang harus dilakukan dalam melakukan evaluasi kromosom, yaitu: evaluasi fungsi objektive (fungsi tujuan) dan konversi fungsi objektive kedalam fungsi fitness. Secara umum, fungsi fitness diturunkan dari fungsi objektive dengan nilai tidak negatif. Apabila ternyata fungsi objektive memiliki nilai negatif, maka perlu ditambahkan suatu konstanta C agar nilai fitness yang terbentuk menjadi tidak negatif.

4. Seleksi

Seleksi ini bertujuan untuk memberikan kesempatan reproduksi yang lebih besar bagi anggota populasi yang paling fit.

5. Operator Genetika

Ada 2 operator genetika, yaitu:

a. Operator untuk melakukan rekombinasi, yang terdiri dari: rekombinasi bernilai biner (*crossover*).
b. Mutasi. mutasi bernilai biner

6. Penentuan parameter

Yang disebut parameter disini adalah parameter kontrol algoritma genetika, yaitu: ukuran populasi (*popsize*), peluang c*rossover* (P_c), dan peluang mutasi (P_m). Nilai parameter ini ditentukan juga berdasarkan permasalahan yang akan dipecahkan. Ada beberapa rekomendasi yang bisa digunakan, antara lain:

a) Untuk permasalahan yang memiliki solusi cukup besar, De Jong merekomendasikan untuk nilai parameter kontrol:
   (popsize; P_c; P_m) = (50; 0,6; 0,001)
b) Bila rata-rata fitness setiap generasi digunakan sebagai indikator, maka Grefenstette merekomendasikan:
   (popsize; P_c; P_m) = (30; 0,95; 0,01)
c) Bila fitness dari individu terbaik dipantau pada setiap generasi, maka usulannya adalah:
   (popsize; P_c; P_m) = (80; 0,45; 0,01)
   Ukuran populasi sebaiknya tidak lebih kecil dari 30, untuk sembarang jenis permasalahan.

2.4.3. Istilah-istilah dalam algoritma genetik

Algoritma genetik [4] merupakan algoritma pencarian yang bekerja berdasarkan mekanisme seleksi alam dan genetika. Pada genetika, kromosom terdiri dari gen-gen. Tiap gen mempunyai sifat tertentu (*allele*), dan posisi tertentu (*locus*). Satu atau lebih kromosom bergabung membentuk paket genetik yang disebut *genotif*. Interaksi *genotif* dengan lingkungannya disebut *fenotif*. Pada algoritma genetik, kromosom berpadanan dengan string dan gen dengan karakter. Setiap karakter mempunyai posisi (*locus*) dan arti tertentu (*allele*). Satu atau lebih string bergabung membentuk struktur (*genotif*), dan apabila struktur tersebut di-*decode*-kan akan diperoleh salah satu alternative solusi (*fenotif*).

Tabel 2.1 Tabel susunan gen ilmu genetik

Ilmu genetik	**Algoritma genetik**
Kromosom	String
Gen	Karakter
Allele	Nilai karakter
Locus	Posisi dalam individu
Genotif	Struktur
Fenotif	parameter

2.4.4. Operator Genetik

Operator genetik [4] digunakan untuk mengkombinasikan (modifikasi) individu dalam aliran populasi untuk menghasilkan individu pada generasi berikutnya. Ada tiga operator genetik yaitu , *crossover* dan mutasi.

2.4.5. *Crossover*

Pada *crossover* [4] akan dipilih secara acak dua individu dan tempat pertukaran, dimana kromosom yang ditandai diantara kedua tempat pertukaran akan bertukar tempat satu sama lain. Proses *crossover* akan membangkitkan offspring baru dengan mengganti sebagian informasi dari parents (orang tua atau induk).

Tujuan dari proses *crossover* adalah untuk menambahkan keanekaragaman individu dalam populasi dengan mengawinkan individu-individu pada populasi sehingga menghasilkan keturunan berupa individu-individu baru untuk ditempatkan pada populasi selanjutnya. Ada beberapa tipe *crossover*, antara lain :

1. *One-cut-point-crossover*

   Pada tipe ini, akan dibuat satu titik *crossover* dimana individu yang dihasilkan akan diambil dari bilangan biner parent pertama dari awal sampai titik *crossover* dan sisanya dari parent kedua seperti pada gambar 2.3. Algoritmanya adalah :

   a. Memilih site secara *random* dari parent pertama
   b. Isi disebelah kanan site pada parent pertama ditukar dengan parent kedua untuk menghasilkan offspring.

   Contoh :

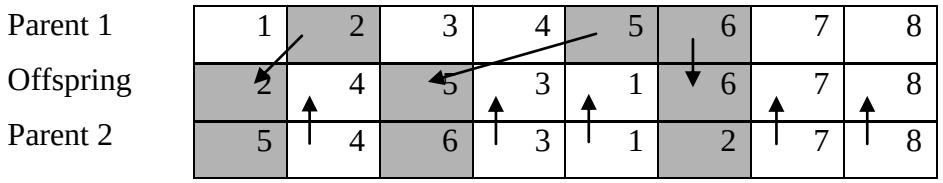
Parent 1	0	1	1	0	1	0	1	1
Parent 2	1	1	0	0	0	1	1	1
Offspring 1	0	1	1	0	0	1	1	1
Offspring 2	1	1	0	0	1	0	1	1

Gambar 2.3. Contoh *single point crossover* [7]

2. *Order-based crossover*

   Pada tipe ini, *offspring* yang dihasilkan hanya satu hasil dari kombinasi kedua parent yang dapat dilihat pada gambar 2.4.

   Contoh :



Gambar 2.4. Contoh *Order Based* C*rossover* [7]

2.4.6. Mutasi

Mutasi [4] menciptakan individu baru dengan melakukan modifikasi satu atau lebih gen dalam individu yang sama. Mutasi berfungsi untuk menggantikan gen yang hilang dari populasi selama proses seleksi serta menyediakan gen yang tidak ada dalam populasi awal. Sehingga mutasi akan meningkatkan variasi populasi. *Shif mutation* diperlihatkan pada gambar 2.5 dan dilakukan dengan cara :

1. Menentukan dua site secara *random*
2. Site pertama ditempatkan ke site kedua, untuk selanjutnya digeser ke kiri seperti terlihat pada gambar berikut.

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

Gambar 2.5. Contoh *Shift Mutation* [7]

2.4.7. Parameter Genetik

Parameter-parameter genetik [4] berguna dalam pengendalian operator-operator genetik. Pemilihan penggunaan nilai-nilai parameter genetik sangat berpengaruh terhadap kinerja algoritma genetik dalam menyelesaikan suatu masalah. Parameter-parameter genetik yang digunakan antara lain :

1. Ukuran populasi

Ukuran popolasi mempengaruhi ukuran efektivitas dan kinerja algoritma genetik. Tidak ada aturan yang pasti tentang berapa nilai ukuran populasi. Apabila ukuran populasi kecil berarti hanya tersedia sedikit pilihan untuk *crossover* dan sebagian kecil dari domain solusi saja yang dieksplorasi untuk setiap generasinya. Sedangkan apabila terlalu besar, kinerja algoritma genetik akan menurun. Penelitian menunjukan ukuran populasi besar tidak mempercepat pencarian solusi. Disarankan ukuran populasi berkisar antara 20 – 30.

2. Jumlah generasi

Jumlah generasi berpengaruh terhadap banyaknya iterasi yang akan dikerjakan dan domain solusi yang akan dieksplorasi untuk setiap generasinya. Semakin besar jumlah generasi berarti semakin banyak iterasi yang dilakukan, dan semakin besar solusi yang dieksplorasi. Sedangkan semakin kecil jumlah generasinya, maka akan semakin kecil iterasi yang akan dilakukan, dan semakin kecil pula solusi yang dieksplorasi untuk tiap generasinya.

3. Probabilitas *crossover* (P_c)

Probabilitas *crossover* akan mengendalikan operator *crossover* dalam setiap generasi dalam populasi yang mengalami *crossover*. Semakin besar nilai probabilitas *crossover*, akan semakin cepat struktur individu baru terbentuk ke dalam populasi. Sedangkan apabila nilai probabilitas *crossover* terlalu besar, individu yang merupakan kandidat solusi terbaik mungkin akan dapat hilang lebih cepat pada generasi selanjutnya. Disarankan nilai probabilitas *crossover* berkisar antara 80 % - 95 %.

4. Probabilitas mutasi (P_m)

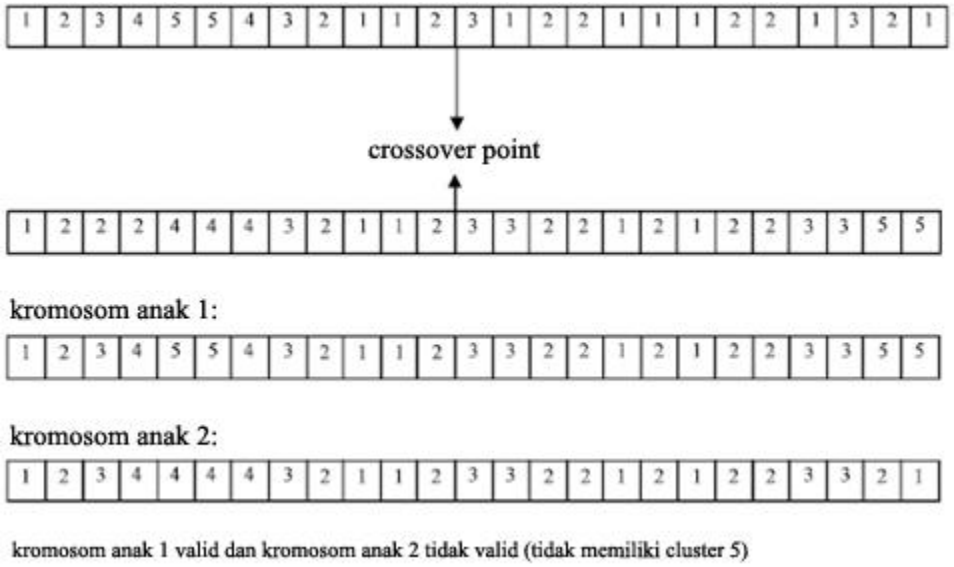
Probabilitas mutasi akan mengendalikan operator mutasi pada setiap generasi. Peluang mutasi yang digunakan biasanya lebih kecil daripada peluang *crossover*. Pada seleksi alam murni, mutasi jarang sekali muncul. Oleh karena itu, operator mutasi pada algoritma genetik juga tidak selalu terjadi. Untuk itulah nilai

peluang mutasi dibuat lebih kecil untuk setiap generasi. Disarankan nilai probabilitas mutasi kecil berkisar antara 0.5 % - 1 %.

2.5. *Fast Genetic K-Means Algorithm*

Fast Genetic K-Means Algorithm (FGKA) [3] merupakan pengembangan dari *Genetic K-means Algorithms* (GKA) yang diusulkan oleh Yi Lu pada tahun 2004. Algoritma ini selalu mampu menghasilkan konvergensi pada global optimal. FGKA mampu menghindari lokal optimal akan tetapi FGKA berjalan 20 kali lebih cepat dibandingkan GKA.

Algoritma GKA sangat baik dalam menemukan optimasi global akan tetapi membutuh waktu yang cukup besar dalam mencapai kondisi konvergensi akibat adanya operator genetika *crossover*. *Crossover* sangat mahal secara biaya komputasi untuk mendapatkan kromosom yang valid (kromosom dengan *cluster* tidak kosong) dan bahkan terkadang dapat menghasilkan kromosom yang tidak valid seperti yang diperlihatkan pada gambar 2.6. Kromosom 1 sebagai hasil *crossover* adalah contoh yang valid karena memiliki setiap *cluster* pada alel-alel nya tetapi kromosom 2 adalah contoh yang tidak valid karena *cluster* 5 tidak memiliki anggota pada kromosom tersebut.

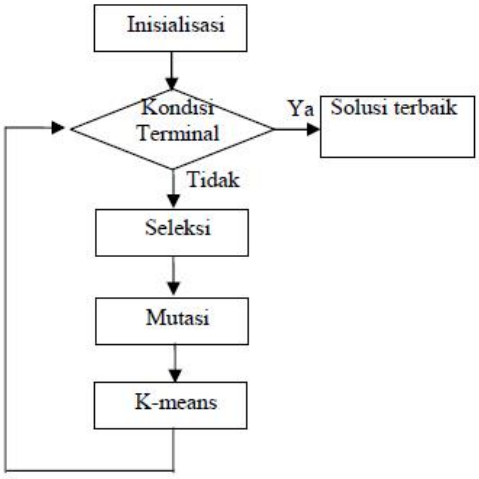


Gambar 2.6. Operator *crossover* pada GKA

FGKA memproses satu populasi yang merupakan kumpulan Z solusi, dimana Z adalah sebuah parameter yang dimasukkan oleh pengguna. Masing-masing solusi dinamakan kromosom, yang dikodekan dengan string a_1 a_2,... a_N dengan panjang N, dimana a_i dinamakan gen yang berisi nilai {1,2,...,K} yang merepresentasikan nomor dari *cluster* yang merupakan memiliki gen tersebut. Sebagai contoh a_1 a_2 a_3 a_4 a_5 = "33212" mengkodekan 5 data dimana data X_1 dan X_2 milik *cluster* 3 dan X_3 dan X_5 milik *cluster* 2 dan X_4 milik *cluster* 1.

String yang tidak legal merepresentasikan sebuah solusi di mana terdapat *cluster* yang tidak mempunyai anggota. Sebagai contoh, jika jumlah K = 3, string a_1 a_2 a_3 a_4 a_5 = "23232" tidak legal karena *cluster* 1 tidak mempunyai anggota. Untuk sebuah kemungkinan solusi Sz= a_1 a_2 ... a_N , maka e(Sz), dinamakan *legality ratio* adalah angka yang menunjukkan jumlah *cluster* yang tidak punya anggota dalam Sz dibagi dengan K. Sehingga kita dapat mengatakan Sz adalah legal jika e(Sz) = 1, selainnya adalah tidak legal.

Pada Gambar 2.3 diilustrasikan diagram alir algoritma FGKA. Algoritma dimulai dari fase inisialisasi, dimana pada fase ini dibangkitkan populasi awal P_0. Populasi dalam generasi berikutnya P_{i+1} didapatkan menggunakan operator genetika dari populasi sebelumnya P_i. Evolusi akan berhenti jika kondisi akhir telah dicapai yaitu pada saat setiap kromosom pada suatu generasi memiliki angka fitness yang konvergen. Operator genetika yang digunakan dalam algoritma FGKA ini adalah : seleksi, mutasi dan operator K-means.



Gambar 2.7. Alir Algoritma FGKA

2.5.1. Inisialisasi

Tahap inisialisasi [3] adalah membangkitkan populasi awal P_0 yang berisi Z solusi, dimana Z adalah parameter yang ditentukan oleh pengguna. Masing-masing gen dalam kromosom menunjukkan nomor *cluster* yang dipilih secara random dengan distribusi uniform dalam jangkauan {1, ..., K}. Dari inisialisasi awal ini dimungkinkan didapatkannya string yang tidak legal. Untuk populasi awal string yang tidak legal dihindari, artinya dalam populasi awal harus didapatkan string yang legal. Apabila string yang tidak legal dapat dimasukkan dalam FGKA, maka sebagian besar solusi yang tidak legal dengan memberikan nilai TWCV yang tinggi dan nilai fitness yang rendah sehingga sebuah solusi yang tidak legal akan mempunyai probabilitas yang rendah untuk bertahan. Kompleksitas dari proses inisialisasi awal ini adalah O(Z).

2.5.2. Nilai *Fitness*

Permasalahan *clustering* [3] terdiri dari N gen yang merepresentasikan N data(obyek). Tujuan dari algoritma FGKA adalah membagi N data ke dalam K *cluster*, dimana *cluster* yang baik adalah *cluster* yang memiliki anggota dengan jarak total yang kecil terhadap masing-masing *centroid cluster*. Itu berarti *cluster* yang memiliki nilai fitness yang tinggi akan memiliki nilai *Total Within-Cluster*

Variation (TWCV) yang rendah dan disebut sebagai masalah minimasi. Untuk mengkonversi masalah minimasi menjadi masalah maksimasi perlu dilakukan *inverse* terhadap TWCV dalam mencari nilai fitness suatu *cluster* dengan rumusan:

$$f(i) = 1/\sum_{j=1}^{n} \begin{matrix}(\text{CON} + \text{difference of allele j in chromosome} \\ \text{i from the corresponding cluster centre})\end{matrix} \quad \text{..... (2.2)}$$

Dimana CON adalah konstanta yang bernilai 1. Konstanta ini disertakan ke dalam persamaan untuk menangani kasus ketika nilai dari TWCV adalah nol. Jika TWCV bernilai nol dan tidak adanya konstanta CON maka nilai fitness akan menjadi ∞ (tidak terhingga).

2.5.3. Seleksi

Seleksi [3] dilakukan dengan maksud untuk menemukan pusat *cluster* global dengan mereduksi nilai dari TWCV setiap kromosom. Pada seleksi dilakukan operator elitism yang merupakan operator yang membiarkan kromosom dengan nilai fitness terbaik selalu dipertahankan.

Kromosom dengan nilai *fitness* terbaik sebagai hasil seleksi oleh operator elitism akan dibandingkan dengan nilai *fitness* kromosom-kromosom generasi berikutnya yang telah mengalami mutasi. Kromosom hasil pengkopian ini akan diurutkan ke dalam serangkaian kromosom pada generasi berikutnya sehingga kromosom dengan nilai fitness terburuk pada generasi berikutnya tidak akan bertahan. Dengan operator elitism ini maka populasi di dalam generasi terbaru akan memiliki serangkaian solusi Z kromosom yang merupakan kandidat-kandidat kromosom dengan nilai fitness terbaik.

2.5.4. Mutasi

Mutasi [3] digunakan untuk menghindari optimum lokal dan membuat pusat *cluster* untuk mendekati optimum global secara menyebar. Karena sifat inisialisasi yang acak, pusat *cluster* tidak akan tepat pada tahap awal sehingga

dilakukan mutasi pada setiap kromosom Z selain kromosom terbaik pada setiap generasi tertentu sesuai dengan frekuensi mutasi yang ditetapkan. Untuk melakukan mutasi perlu dilakukannya pemeriksaan probabilitas dari suatu data X_{ij} pada kromosom Z_i yang didapatkan secara *random*.

$$D_m(X_{ij}) = (\text{abs}\ (X_{ij} - C_j) - \text{tot_min})$$
$$P_m(X_{ij}) = 0 \text{ if } D_m = 0$$
$$P_m(X_{ij}) = 1 \text{ otherwise} \qquad \text{..... (2.3)}$$

Dimana tot_min merupakan jarak terkecil antara alel yang berada dalam *cluster* data X_{ij} dan pusat *cluster* data X_{ij}.

Jika P_m adalah 1 maka nilai dari alel terpilih akan dipetakan terhadap pusat *cluster* yang secara acak dipilih. Kemudian jika P_m adalah 0 maka kromosom tersebut tidak perlu dilakukan mutasi.

Mutasi ini didefinisikan sebagai berikut:

1. X_{ij} dipilih secara random, hal ini bertujuan untuk mencapai optimal pada wilayah global.
2. Probabilitas dari X_{ij} akan besar ketika jarak antara alel tersebut dengan pusat *cluster* adalah besar. Ini dimaksudkan agar pusat *cluster* baru yang terbentuk akibat mutasi tidak berjarak terlalu berdekatan atau bahkan merupakan nilai dari pusat *cluster* yang sudah eksis.

2.6. Operator Sobel

Operator Sobel [8] merupakan sebuah teknik deteksi tepi yang sederhana dan memiliki tingkat komputasi yang cepat dan akurat. Pada umumnya operator ini digunakan untuk citra *grayscale*. Operator Sobel dapat digambarkan dengan dua buah matriks berukuran 3 x 3 seperti pada gambar 2.8.

$$\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \quad \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$
$$G_x \qquad G_y$$

Gambar 2.8. Matriks operator sobel

Matriks operator sobel dapat merespon tepian maksimal hingga 45º. Kedua matriks tersebut dapat diterapkan pada citra secara terpisah, baik untuk mendapatkan tepian horizontal maupun tepian vertikal. Kedua matriks ini juga dapat dikombinasikan untuk mendapatkan hasil *gradient* gabungan dengan menggunakan persamaan :

$$|G| = \sqrt{G_x^2 + G_y^2} \quad \text{.....(2.4)}$$

atau dengan persamaan

$$|G| = |G_x| + |G_y| \quad \text{.....(2.5)}$$

2.7. Konsep Dasar Analisis Sistem

Adapun konser dasar analisis sistem adalah sebagai berikut:

2.7.1. OOP (*Object Oriented Programming*)

OOP (*Object Oriented Programming*) atau yang dikenal dengan Pemrograman Berorientasi Objek merupakan paradigma pemrograman yang berorientasikan kepada objek. Semua data dan fungsi di dalam paradigma ini dibungkus ke dalam kelas-kelas atau objek-objek. Model data berorientasi objek dikatakan dapat memberi fleksibilitas yang lebih, kemudahan mengubah program, dan digunakan luas dalam teknik piranti lunak skala besar. Lebih jauh lagi, pendukung OOP mengklaim bahwa OOP lebih mudah dipelajari bagi pemula dibanding dengan pendekatan sebelumnya, dan pendekatan OOP lebih mudah dikembangkan dan dirawat.

Dengan menggunakan OOP maka dalam melakukan pemecahan suatu masalah kita tidak melihat bagaimana cara menyelesaikan suatu masalah tersebut (terstruktur) tetapi objek-objek apa yang dapat melakukan pemecahan masalah tersebut. Sebagai contoh anggap kita memiliki sebuah departemen yang memiliki manager, sekretaris, petugas administrasi data dan lainnya. Misal manager tersebut ingin memperoleh data dari bag administrasi maka manager tersebut tidak harus mengambilnya langsung tetapi dapat menyuruh petugas bagian administrasi

untuk mengambilnya. Pada kasus tersebut seorang manager tidak harus mengetahui bagaimana cara mengambil data tersebut tetapi manager bisa mendapatkan data tersebut melalui objek petugas administrasi. Jadi untuk menyelesaikan suatu masalah dengan kolaborasi antar objek-objek yang ada karena setiap objek memiliki deskripsi tugasnya sendiri. Pemrograman orientasi-objek menekankan konsep berikut:

2.7.2. Kelas (*Class*)

Kelas [6] merupakan kumpulan atas definisi data dan fungsi-fungsi dalam suatu unit untuk suatu tujuan tertentu. Sebagai contoh '*Class of dog*' adalah suatu unit yang terdiri atas definisi-definisi data dan fungsi-fungsi yang menunjuk pada berbagai macam perilaku/turunan dari anjing. Sebuah *Class* adalah dasar dari modularitas dan struktur dalam pemrograman berorientasi objek. Sebuah *Class* secara tipikal sebaiknya dapat dikenali oleh seorang non-programmer sekalipun terkait dengan domain permasalahan yang ada, dan kode yang terdapat dalam sebuah *Class* sebaiknya (relatif) bersifat mandiri dan independen (sebagaimana kode tersebut digunakan jika tidak menggunakan OOP). Dengan modularitas, struktur dari sebuah program akan terkait dengan aspek-aspek dalam masalah yang akan diselesaikan melalui program tersebut. Cara seperti ini akan menyederhanakan pemetaan dari masalah ke sebuah program ataupun sebaliknya.

2.7.3. Objek (*Object*)

Objek [6] membungkus data dan fungsi bersama menjadi suatu unit dalam sebuah program komputer. Objek merupakan dasar dari modularitas dan struktur dalam sebuah program komputer berorientasi objek.

2.7.4. Abstraksi (*Abstract*)

Abstraksi [6] merupakan kemampuan sebuah program untuk melewati aspek informasi yang diproses olehnya, yaitu kemampuan untuk memfokus pada inti. Setiap objek dalam sistem melayani sebagai model dari "pelaku" abstrak yang dapat melakukan kerja, laporan dan perubahan keadaannya, dan

berkomunikasi dengan objek lainnya dalam sistem, tanpa mengungkapkan bagaimana kelebihan ini diterapkan. Proses, fungsi atau metode dapat juga dibuat abstrak, dan beberapa teknik digunakan untuk mengembangkan sebuah pengabstrakan.

2.7.5. Enkapsulasi (*Encapsulation*)

Enkapsulasi [6] memastikan pengguna sebuah objek tidak dapat mengganti keadaan dalam dari sebuah objek dengan cara yang tidak layak. Hanya metode dalam objek tersebut yang diberi ijin untuk mengakses keadaannya. Setiap objek mengakses interface yang menyebutkan bagaimana objek lainnya dapat berinteraksi dengannya. Objek lainnya tidak akan mengetahui dan tergantung kepada representasi dalam objek tersebut.

2.7.6. Polimorfisme (*Polymorphism*)

Polimorfisme [6] tidak bergantung kepada pemanggilan subrutin, bahasa orientasi objek dapat mengirim pesan. Metode tertentu yang berhubungan dengan sebuah pengiriman pesan tergantung kepada objek tertentu di mana pesa tersebut dikirim. Contohnya, bila sebuah burung menerima pesan "gerak cepat", dia akan menggerakan sayapnya dan terbang. Bila seekor singa menerima pesan yang sama, dia akan menggerakkan kakinya dan berlari. Keduanya menjawab sebuah pesan yang sama, namun yang sesuai dengan kemampuan hewan tersebut. Ini disebut polimorfisme karena sebuah variabel tunggal dalam program dapat memegang berbagai jenis objek yang berbeda selagi program berjalan, dan teks program yang sama dapat memanggil beberapa metode yang berbeda di saat yang berbeda dalam pemanggilan yang sama. Hal ini berlawanan dengan bahasa fungsional yang mencapai polimorfisme melalui penggunaan fungsi kelas-pertama.

2.7.7. Inheritans (*Inheritance*)

Inheritans [6] mengatur polimorfisme dan enkapsulasi dengan mengijinkan objek didefinisikan dan diciptakan dengan jenis khusus dari objek

yang sudah ada objek-objek ini dapat membagi (dan memperluas) perilaku mereka tanpa harus mengimplementasi ulang perilaku tersebut (bahasa berbasis-objek tidak selalu memiliki inheritas).

2.7.8. UML (*Unified Modelling Language*)

UML (*Unified Modeling Language*) [6] adalah bahasa spesifikasi standar untuk mendokumentasikan, menspesifikasikan, dan membangun sistem. Unified Modeling Language (UML) adalah himpunan struktur dan teknik untuk pemodelan desain program berorientasi objek (OOP) serta aplikasinya. UML adalah metodologi untuk mengembangkan sistem OOP dan sekelompok perangkat *tool* untuk mendukung pengembangan sistem tersebut. UML mulai diperkenalkan oleh *Object Management Group*, sebuah organisasi yang telah mengembangkan model, teknologi, dan standar OOP sejak tahun 1980-an. Sekarang UML sudah mulai banyak digunakan oleh para praktisi OOP.

UML merupakan dasar bagi perangkat (*tool*) desain berorientasi objek dari IBM. UML adalah suatu bahasa yang digunakan untuk menentukan, memvisualisasikan, membangun, dan mendokumentasikan suatu sistem informasi. 25 UML dikembangkan sebagai suatu alat untuk analisis dan desain berorientasi objek oleh Grady Booch, Jim Rumbaugh, dan Ivar Jacobson. Namun demikian UML dapat digunakan untuk memahami dan mendokumentasikan setiap sistem informasi. Penggunaan UML dalam industri terus meningkat. Ini merupakan standar terbuka yang menjadikannya sebagai bahasa pemodelan yang umum dalam industri peranti lunak dan pengembangan system. UML menyediakan 10 macam diagram untuk memodelkan aplikasi berorientasi objek, yaitu:

2.7.9. *Use Case Diagram*

Use case diagram [6] adalah gambaran graphical dari beberapa atau semua Aktor, use-case, dan interaksi diantara komponen-komponen tersebut yang memperkenalkan suatu sistem yang akan dibangun. Use-case diagram menjelaskan manfaat suatu sistem jika dilihat menurut pandangan orang yang

berada di luar sistem. Diagram ini menunjukkan fungsionalitas suatu sistem atau kelas dan bagaimana sistem tersebut berinteraksi dengan dunia luar.

Use-case diagram dapat digunakan selama proses analisis untuk menangkap requirement sistem dan untuk memahami bagaimana sistem seharusnya bekerja. Selama tahap desain, *use case diagram* berperan untuk menetapkan perilaku (*behavior*) sistem saat diimplementasikan. Dalam sebuah model mungkin terdapat satu atau beberapa use-case diagram. Kebutuhan atau *requirements system* adalah fungsionalitas apa yang harus disediakan oleh sistem kemudian didokumentasikan pada model use-case yang menggambarkan fungsi sistem yang diharapkan (*use case*), dan yang mengelilinginya (aktor), serta hubungan antara aktor dengan use case (*use case diagram*) itu sendiri.

2.7.10. *Conceptual Diagram*

Sebuah diagram konseptual [6] merupakan representasi visual dari cara di mana konsep-konsep abstrak terkait. Hal ini digunakan sebagai bantuan dalam memvisualisasikan proses atau sistem tingkat tinggi melalui serangkaian garis yang unik dan bagan. Diagram konseptual secara luas digunakan dalam segala bidang seperti bisnis, ilmu pengetahuan, dan manufaktur.

2.7.11. *Sequence Diagram*

Sequence Diagram [6] menggambarkan interaksi antar objek di dalam dan di sekitar sistem (termasuk pengguna, display, dan sebagainya) berupa message yang digambarkan terhadap waktu. *Sequence Diagram* terdiri atar dimensi vertikal (waktu) dan dimensi horizontal (objek-objek yang terkait).

Sequence Diagram biasa digunakan untuk menggambarkan skenario atau rangkaian langkah-langkah yang dilakukan sebagai respons dari sebuah event untuk menghasilkan output tertentu. Diawali dari apa yang *men-trigger* aktivitas tersebut, proses dan perubahan apa saja yang terjadi secara internal dan output apa yang dihasilkan.

2.7.12. *Collaboration Diagram*

Collaboration diagram [6] yaitu diagram yang mengelompokkan pesan pada kumpulan diagram sekuen menjadi sebuah diagram. Dalam diagram tersebut terdapat method yang dijalankan antara objek yang satu dan objek lainnya. Di diagram kolaborasi ini, objek harus melakukan sinkronisasi pesan dengan serangkaian pesan-pesan lainnya. *Collaboration Diagram* lebih menekankan kepada peran setiap objek dan bukan pada waktu penyampaian pesan.

2.7.13. *State Diagram*

State Diagram [6] adalah diagram untuk menggambarkan behavior, yaitu perubahan state di suatu *Class* berdasarkan event dan pesan yang dikirimkan dan diterima oleh *Class* tersebut. Setiap diagram state hanya boleh memiliki satu start state (*initial state*) dan boleh memiliki satu atau lebih dari satu stop states (*final state*).

2.7.14. *Activity Diagram*

Activity Diagram [6] memiliki pengertian yaitu lebih fokus kepada menggambarkan proses bisnis dan urutan aktivitas dalam sebuah proses. Dipakai pada business modeling untuk memperlihatkan urutan aktifitas proses bisnis. Memiliki struktur diagram yang mirip *flowchart* atau data *flow diagram* pada perancangan terstruktur. Memiliki pula manfaat yaitu apabila kita membuat diagram ini terlebih dahulu dalam memodelkan sebuah proses untuk membantu memahami proses secara keseluruhan. Dan activity dibuat berdasarkan sebuah atau beberapa *use case* pada *use case diagram*.

2.7.15. *Class Diagram*

Class diagram [6] adalah sebuah *Class* yang menggambarkan struktur dan penjelasan *Class*, paket, dan objek serta hubungan satu sama lain seperti *containment*, pewarisan, asosiasi, dan lain-lain. *Class diagram* juga menjelaskan hubungan antar *Class* dalam sebuah sistem yang sedang dibuat dan bagaimana caranya agar mereka saling berkolaborasi untuk mencapai sebuah tujuan.

2.7.16. *Object Diagram*

Objek diagram [6] adalah diagram yang memberikan gambaran struktur model sebuah sistem, dalam kurun waktu tertentu. Diagram objek yang berasal dari diagram kelas sehingga diagram objek tergantung pada diagram kelas. Objek Diagram, kadang-kadang disebut sebagai *diagram instance* sangat mirip dengan diagram kelas. Seperti diagram kelas *object diagram* juga menunjukkan hubungan antara obyek, tetapi *object diagram* menggunakan contoh-contoh dunia nyata. *Object diagram* digunakan untuk menunjukkan bagaimana sistem akan terlihat seperti pada waktu tertentu. Karena ada data yang tersedia di objek-objek diagram sering digunakan untuk menjelaskan hubungan yang kompleks antara objek.

2.7.17. *Component Diagram*

Component diagram [6] adalah diagram UML yang menampilkan komponen dalam system dan hubungan antara mereka. Pada component *View*, akan difokuskan pada organisasi fisik system. Pertama, diputuskan bagaimana kelas-kelas akan diorganisasikan menjadi kode pustaka. Kemudian akan dilihat bagaimana perbedaan antara berkas eksekusi, berkas *dynamic link library* (DDL), dan berkas runtime lainnya dalam system.

2.7.18. *Deployment Diagram*

Deployment Diagram [6] adalah diagram yang menggambarkan detail bagaimana komponen disebar kedalam infrastruktur sistem, dimana komponen akan terletak (pada mesin, node, server atau piranti keras apa), bagaimana kemampuan jaringan pada lokasi tersebut, spesifikasi server, dan hal-hal lain yang bersifat fisikal.

2.8. IDE yang Digunakan

IDE atau singkatan dari (*Integrated Development Environment*) adalah program komputer yang memiliki beberapa fasilitas yang diperlukan dalam pembangunan perangkat lunak. Tujuan dari IDE adalah untuk menyediakan semua utilitas yang diperlukan dalam membangun perangkat lunak.

Sebuah IDE, atau secara bebas dapat diterjemahkan sebagai Lingkungan Pengembangan Terpadu, setidaknya memiliki fasilitas:

a. Editor, yaitu fasilitas untuk menuliskan kode sumber dari perangkat lunak.
b. Compiler, yaitu fasilitas untuk mengecek sintaks dari kode sumber kemudian mengubah dalam bentuk binari yang sesuai dengan bahasa mesin.
c. Linker, yaitu fasilitas untuk menyatukan data binari yang beberapa kode sumber yang dihasilkan compiler sehingga data-data binari tersebut menjadi satu kesatuan dan menjadi suatu program komputer yang siap dieksekusi.
d. Debuger, yaitu fasilitas untuk mengetes jalannya program, untuk mencari bug/kesalahan yang terdapat dalam program.

Sampai tahap tertentu IDE modern dapat membantu memberikan saran yang mempercepat penulisan. Pada saat penulisan kode, IDE juga dapat menunjukan bagian-bagian yang jelas mengandung kesalahan atau keraguan.

2.8.1. Microsoft Visual Studio

Microsoft Visual Studio [7] merupakan sebuah perangkat lunak lengkap (*suite*) yang dapat digunakan untuk melakukan pengembangan aplikasi, baik itu aplikasi bisnis, aplikasi personal, ataupun komponen aplikasinya, dalam bentuk aplikasi console, aplikasi Windows, ataupun aplikasi Web. Visual Studio mencakup kompiler, SDK, Integrated Development Environment (IDE), dan dokumentasi (umumnya berupa MSDN Library). Kompiler yang dimasukkan ke dalam paket Visual Studio antara lain Visual C++, Visual C#, Visual Basic, Visual Basic .NET, Visual InterDev, Visual J++, Visual J#, Visual FoxPro, dan Visual SourceSafe.

Microsoft Visual Studio dapat digunakan untuk mengembangkan aplikasi dalam *native code* (dalam bentuk bahasa mesin yang berjalan di atas Windows) ataupun *managed code* (dalam bentuk Microsoft *Intermediate Language* di atas .NET *Framework*). Selain itu, Visual Studio juga dapat digunakan untuk mengembangkan aplikasi Silverlight, aplikasi Windows Mobile (yang berjalan di atas .NET *Compact Framework*).

Visual Studio kini telah menginjak versi Visual Studio 9.0.21022.08, atau dikenal dengan sebutan Microsoft Visual Studio 2008 yang diluncurkan pada 19 November 2007, yang ditujukan untuk platform Microsoft .NET *Framework* 3.5. Versi sebelumnya, Visual Studio 2005 ditujukan untuk platform .NET *Framework* 2.0 dan 3.0. Visual Studio 2003 ditujukan untuk .NET *Framework* 1.1, dan Visual Studio 2002 ditujukan untuk .NET *Framework* 1.0. Versi-versi tersebut di atas kini dikenal dengan sebutan Visual Studio .NET, karena memang membutuhkan Microsoft .NET *Framework*. Sementara itu, sebelum muncul Visual Studio .NET, terdapat Microsoft Visual Studio 6.0 (VS1998).