

BAB 1

PENDAHULUAN

1.1 Latar Belakang

Pengelompokan (*clustering*) merupakan prosedur untuk mencari struktur alami dari suatu kumpulan data. Proses ini melibatkan pemilihan data atau objek ke dalam kelompok (*cluster*) sehingga objek-objek dalam cluster yang sama akan lebih mirip satu sama lain dibandingkan dengan objek yang berada di *cluster* lain. *Clustering* berguna untuk mereduksi data (mereduksi data dengan volume besar ke dalam kelompok-kelompok dengan karakteristik tertentu), mengembangkan skema klasifikasi (juga dikenal sebagai taksonomi), dan memberikan masukan atau dukungan terhadap hipotesis mengenai struktur suatu data.

Clustering merupakan salah satu teknik pembelajaran tak terarah (*unsupervised learning*). Pembagian kelompok dalam *clustering* tidak berdasarkan sesuatu yang telah diketahui sebelumnya, melainkan berdasarkan kesamaan tertentu menurut suatu ukuran tertentu [1].

Salah satu algoritma pengelompokan yang paling sering digunakan adalah *K-means* yang dilakukan dengan cara membagi data ke dalam K kelompok. Kelompok tersebut dibentuk dengan cara meminimalkan jarak antara titik pusat *cluster* (*centroid*) dengan setiap anggota *cluster* tersebut. Titik pusat *cluster* dicari dengan menggunakan rata-rata (*mean*) dari nilai setiap anggota *cluster*. Dalam hal ini, setiap anggota *cluster* dimodelkan sebagai vektor dalam n dimensi (n merupakan banyaknya atribut). *K-means* sudah terbukti efektif dalam melakukan pengelompokan dalam situasi apapun. Namun, cara tersebut tetap saja memiliki kekurangan yaitu dapat terjebak dalam *local optima* tergantung dengan pemilihan *centroid* awal [2].

Masalah *local optima* dapat ditangani menggunakan *Genetic Algorithm* (GA) yang telah terbukti efektif dalam menyelesaikan masalah pencarian dan optimasi. GA merupakan teknik pencarian heuristik tingkat tinggi yang menirukan proses evolusi yang secara alami terjadi [3] berdasarkan prinsip *survival of the fittest*. Algoritma ini dinamakan demikian karena menggunakan konsep-konsep dalam genetika sebagai model pemecahan masalahnya [4].

Dalam GA, parameter dari *search space* dikodekan dalam bentuk deretan objek yang disebut kromosom. Kumpulan kromosom tersebut lalu dikenal sebagai populasi. Pada awalnya, populasi dibangkitkan secara acak. Kemudian, akan dipilih beberapa kromosom menggunakan teknik *roulette wheel selection* berdasarkan fungsi *fitness*. Operasi dasar yang terinspirasi dari Ilmu Biologi seperti persilangan (*crossover*) dan mutasi (*mutation*) digunakan untuk membangkitkan generasi berikutnya. Proses seleksi, persilangan, dan mutasi ini berlangsung dalam jumlah generasi tertentu atau sampai kondisi akhir tercapai.

Fungsi *fitness* tidak hanya berfungsi untuk menentukan seberapa baik solusi yang dihasilkan namun juga menentukan seberapa dekat solusi tersebut dengan hasil yang optimal [4]. Oleh karena itu, diperlukan fungsi *fitness* yang cocok sehingga GA dapat menghasilkan keluaran yang optimal. Pada masalah *clustering* menggunakan GA, maka fungsi *fitness* yang digunakan harus bisa menggambarkan bahwa seluruh elemen sudah berada dalam *cluster* yang terbaik dan sudah sesuai.

1.2 Rumusan Masalah

Berdasarkan latar belakang yang telah dipaparkan, rumusan masalah dari penelitian ini adalah sebagai berikut:

1. Bagaimana algoritma genetik dapat digunakan untuk mengelompokkan dokumen?
2. Bagaimana membangun perangkat lunak yang menggunakan algoritma genetik untuk dapat mengelompokkan dokumen?

1.3 Tujuan Penelitian

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

1. Mempelajari algoritma genetik dan hubungannya dengan pengelompokan dokumen.
2. Membangun perangkat lunak yang mengimplementasikan algoritma genetik untuk dapat mengelompokkan dokumen.

1.4 Batasan Masalah

Rumusan masalah yang telah disebutkan memiliki ruang lingkup yang cukup luas. Dengan menyadari terbatasnya waktu serta kemampuan, penelitian ini akan difokuskan dengan memperlihatkan batasan masalah sebagai berikut:

1. Jenis dokumen yang dapat diproses dengan perangkat lunak yang akan dibuat hanyalah *Text Document* dengan ekstensi *TXT*.
2. Informasi dari dokumen yang akan diproses dalam pengelompokan hanya berasal dari teks yang menjadi isi dari dokumen tersebut. Gambar dan *metadata* (pemilik, tanggal modifikasi) tidak akan diperhitungkan.

1.5 Metodologi

Langkah-langkah yang akan dilakukan dalam penelitian ini adalah:

1. Melakukan studi literatur mengenai model ruang vektor, *Document Clustering* (pengelompokan dokumen), *Genetic Algorithm* (algoritma genetik), dan penggunaan algoritma genetik dalam pengelompokan dokumen.
2. Mencari dokumen yang akan dijadikan *training* dan *test datasets*.
3. Membuat rancangan perangkat lunak yang menggunakan algoritma genetik sebagai algoritma pengelompokan dokumen.
4. Mengimplementasikan hasil rancangan menjadi perangkat lunak dalam bahasa pemrograman Java.
5. Melatih dan menguji perangkat lunak dengan dokumen yang telah tersedia.
6. Mengevaluasi hasil pengujian lalu lakukan implementasi dan pengujian kembali sampai didapatkan hasil yang sudah sesuai dengan harapan.

1.6 Sistematika Pembahasan

Dokumentasi dari penelitian ini akan disajikan dalam enam bab dengan sistematika pembahasan sebagai berikut:

1. Bab 1 Pendahuluan

Bab 1 berisi latar belakang pemilihan "Pengelompokan Dokumen berbasis Algoritma Genetika" sebagai judul dari penelitian ini. Selain itu, dibahas juga rumusan masalah, tujuan penelitian, batasan masalah, serta metodologi penelitian yang menjadi acuan dari penelitian ini.

2. Bab 2 Dasar Teori

Bab 2 memuat landasan teori yang digunakan dalam penelitian ini. Konsep-konsep yang dibahas yaitu temu kembali informasi, pembelajaran mesin, algoritma genetika, dan GA dalam pengelompokan.

3. Bab 3 Analisis

4. Bab 4 Perancangan

5. Bab 5 Implementasi dan Pengujian

6. Bab 6 Kesimpulan

BAB 2

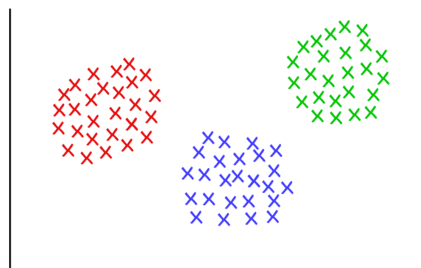
LANDASAN TEORI

Pengelompokan dokumen berkaitan erat dengan dua bidang ilmu dalam informatika. Pengelompokan dalam informatika merupakan bagian dari bidang pembelajaran mesin. Terdapat dua jenis pengelompokan dalam pembelajaran mesin yaitu *clustering* dan *classification*. *Clustering* merupakan salah satu jenis pembelajaran tak terarah (*unsupervised learning*) karena setiap elemen dikelompokkan berdasarkan karakteristik dari elemen tersebut. Sedangkan *classification* merupakan jenis pembelajaran terarah (*supervised learning*) karena setiap elemen dikelompokkan berdasarkan label yang telah ditentukan sebelumnya. Pada penelitian ini, jenis pengelompokan yang akan digunakan adalah *clustering*.

2.1 Pengelompokan

2.1.1 Definisi pengelompokan

Pengelompokan (*clustering*) merupakan sebuah metode untuk menggabungkan himpunan objek ke dalam kelompok-kelompok sedemikian rupa sehingga objek dalam kelompok (*cluster*) lebih mirip (karena suatu hal) satu sama lain daripada objek di kelompok lain [5]. Pengelompokan seringkali tertukar dengan istilah klasifikasi yang hanya bertugas untuk memisahkan objek ke dalam kelas-kelas yang telah ditentukan sebelumnya. Masukan dari proses pengelompokan adalah kumpulan objek dan banyaknya kelompok (*cluster*) yang akan dibentuk. Keluaran yang dihasilkan dari proses pengelompokan adalah kelompok objek yang telah dibentuk beserta anggotanya. Setiap objek dikelompokkan berdasarkan kesamaan tertentu. Sebagai ilustrasi dari pengelompokan (Gambar 2.1), terdapat tiga *cluster* yang ditandai dengan warna merah, biru, dan hijau. Objek-objek yang berwarna sama dianggap mirip sehingga dimasukkan ke dalam kelompok yang sama. Begitu juga dengan objek yang berbeda warna dianggap tidak mirip sehingga perlu dipisahkan.



Gambar 2.1: Contoh *cluster* hasil pengelompokan

2.1.2 Aplikasi Pengelompokan

Pengelompokan memegang peran penting dalam beberapa bidang seperti *recommender system* dan penambangan data. Berikut adalah penjelasan singkat aplikasi pengelompokan dalam setiap bidang yang telah disebutkan.

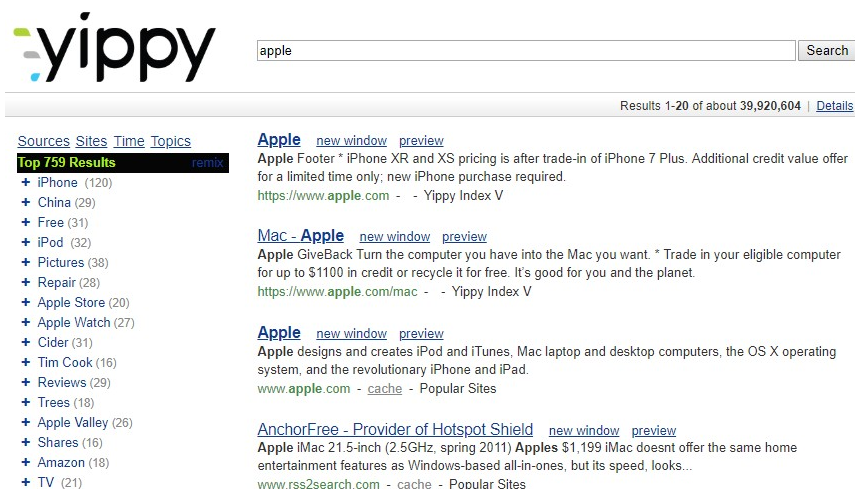
Recommender System

Recommender system adalah suatu sistem yang berfungsi untuk memprediksikan keinginan pengguna berdasarkan masukan yang diberikan oleh pengguna [6]. Ada dua jenis sistem rekomendasi yaitu *Content-based Recommendation* dan *Collaborative Filtering*.

- *Content-based Recommendation*: mempelajari apa yang pengguna sukai lalu mencari objek lain yang mungkin juga disukai oleh pengguna tersebut berdasarkan apa yang disukainya. Pencarian ini dilakukan dengan mengusulkan objek-objek yang berada dalam kelompok (*cluster*) yang sama dengan objek yang disukai oleh pengguna. Hal ini dilakukan dengan asumsi pengguna akan menyukai barang yang mirip dengan barang yang disukai.
- *Collaborative Filtering*: memberikan usulan berdasarkan apa yang disukai pengguna yang serupa dengan seseorang dengan mengasumsikan jika pengguna serupa menyukai suatu objek, maka orang tersebut akan menyukai objek yang sama. Pengguna serupa didapatkan dengan mencari orang-orang yang berada dalam satu *cluster* dengan memperhitungkan atribut dari pengguna (usia, jenis kelamin, tempat domisili, hobi, dll).

Search Result Clustering

Salah satu kegunaan pengelompokan dalam bidang penambangan data adalah untuk mengelompokkan hasil pencarian (*search result clustering*) [7]. Setiap kata kunci dalam sebuah pencarian mungkin dapat masuk ke dalam berbagai kategori. Misalkan kata kunci "apple" dapat berarti buah apel atau perusahaan teknologi *apple*. Dengan menggunakan pengelompokan hasil pencarian, maka hasil dari pencarian akan dimasukkan ke dalam kelompok-kelompok topik dan pengguna dapat memilih topik mana yang dimaksud untuk dapat mengeluarkan hasil yang lebih spesifik. Mesin pencari yang mengelompokkan hasil pencariannya dinamakan dengan *clustering search engine*. Salah satu contoh *clustering search engine* adalah Yippy¹.



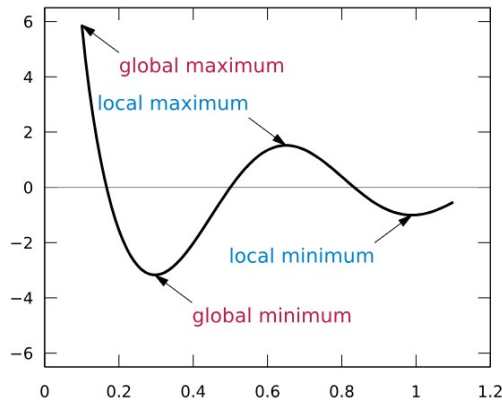
Gambar 2.2: Hasil pencarian (*clustered search result*) di Yippy

¹<https://yippy.com/>

Hasil pencarian di Yippy dengan kata kunci "apple" ditunjukkan dalam Gambar 2.2. Bagian sebelah kiri pada Gambar 2.2 merupakan kelompok-kelompok kategori dari kata kunci yang dimasukkan sehingga pengguna dapat memilih kategori yang sesuai dengan yang mereka maksud. "iPhone", "China", "Free", dan seterusnya merupakan kelompok yang dihasilkan apabila pengguna memasukkan kata kunci "apple".

2.1.3 Local Optimum

Local optimum adalah suatu solusi yang optimal (baik maksimal maupun minimal) diantara kandidat solusi yang berdekatan dalam masalah optimasi. Dikatakan lokal karena solusi ini hanya optimal apabila dibandingkan dengan kandidat solusi yang berdekatan, tidak optimal secara keseluruhan (*global optimum*). Contoh dari *local* dan *global* optimum ditunjukkan pada Gambar 2.3. *Local minimum* memiliki nilai yang paling kecil apabila dibandingkan dengan nilai-nilai lain yang berdekatan dengannya, begitu pula dengan *local maximum*. Sedangkan *global minimum* dan *global maximum* memiliki nilai yang paling minimum dan maksimum dalam keseluruhan himpunan kandidat solusi.



Gambar 2.3: Local Optimum dan Global Optimum

Suatu program optimasi dapat terjebak di *local optimum*. Sebagai contoh pada Gambar 2.3, apabila suatu program mencari solusi yang merupakan nilai maksimum maka program dapat terjebak pada nilai 2 yang merupakan *local maximum* karena seharusnya keluaran dari program tersebut adalah 6 yang merupakan *global maximum*. Sedangkan apabila suatu program mencari solusi berupa nilai minimum maka program dapat terjebak pada nilai -1 yang merupakan *local minimum* karena seharusnya keluaran dari program tersebut adalah -3 yang merupakan *global minimum*.

2.2 K-Means

K-means merupakan salah satu algoritma pengelompokan yang umum digunakan saat ini. Algoritma ini membagi objek ke dalam K cluster. Setiap cluster direpresentasikan dengan titik tengahnya (*centroid*). Titik tengah akan dihitung sebagai rata-rata dari semua titik objek dari cluster tersebut dalam setiap iterasinya. Persamaan 2.1 merupakan persamaan untuk menghitung *centroid*

$$\mu_i = \frac{1}{N_i} \sum_{q=1}^{N_i} x_q \quad (2.1)$$

dengan μ_i merupakan *centroid* ke- i , N_i merupakan jumlah titik objek pada cluster ke- i , dan x_q merupakan titik ke- q pada cluster ke- i .

Algorithm 1 K-Means**Input:** S (himpunan titik objek), K (Jumlah *cluster*)**Output:** himpunan *cluster*

- 1: Pilih K titik objek sebagai himpunan awal *centroid*.
- 2: **repeat**
- 3: Bentuk K *cluster* dengan menempatkan setiap titik objek ke *cluster* dengan *centroid* terdekat.
- 4: Hitung ulang *centroid* untuk setiap *cluster*.
- 5: **until** *Centroid* tidak berubah.

1 Baris ke-4 pada Algoritma 1 dilakukan dengan cara menentukan *centroid* baru yang dihitung
 2 menggunakan Persamaan 2.1. Baris ke-5 pada Algoritma 1 berarti *centroid* mengalami pergeseran
 3 yang tidak terlalu signifikan (lebih kecil dari suatu nilai ϵ yang merupakan masukan dari pengguna).

2.3 Algoritma Genetika

5 Algoritma genetika atau biasa disebut *Genetic Algorithm*(GA) adalah suatu algoritma pencarian
 6 yang terinspirasi dari proses seleksi alam yang terjadi secara alami dalam proses evolusi. Di alam,
 7 individu dalam suatu populasi berkompetisi satu sama lain untuk memperebutkan tempat tinggal,
 8 makanan, dll [4]. Bahkan setiap individu dalam spesies yang sama pun harus bersaing menarik
 9 lawan jenis untuk berkembang biak. Individu yang kurang baik akan memiliki peluang bertahan
 10 hidup lebih kecil, dan individu yang bisa beradaptasi dengan baik atau "*fit*" akan menghasilkan
 11 keturunan dengan jumlah yg relatif banyak.

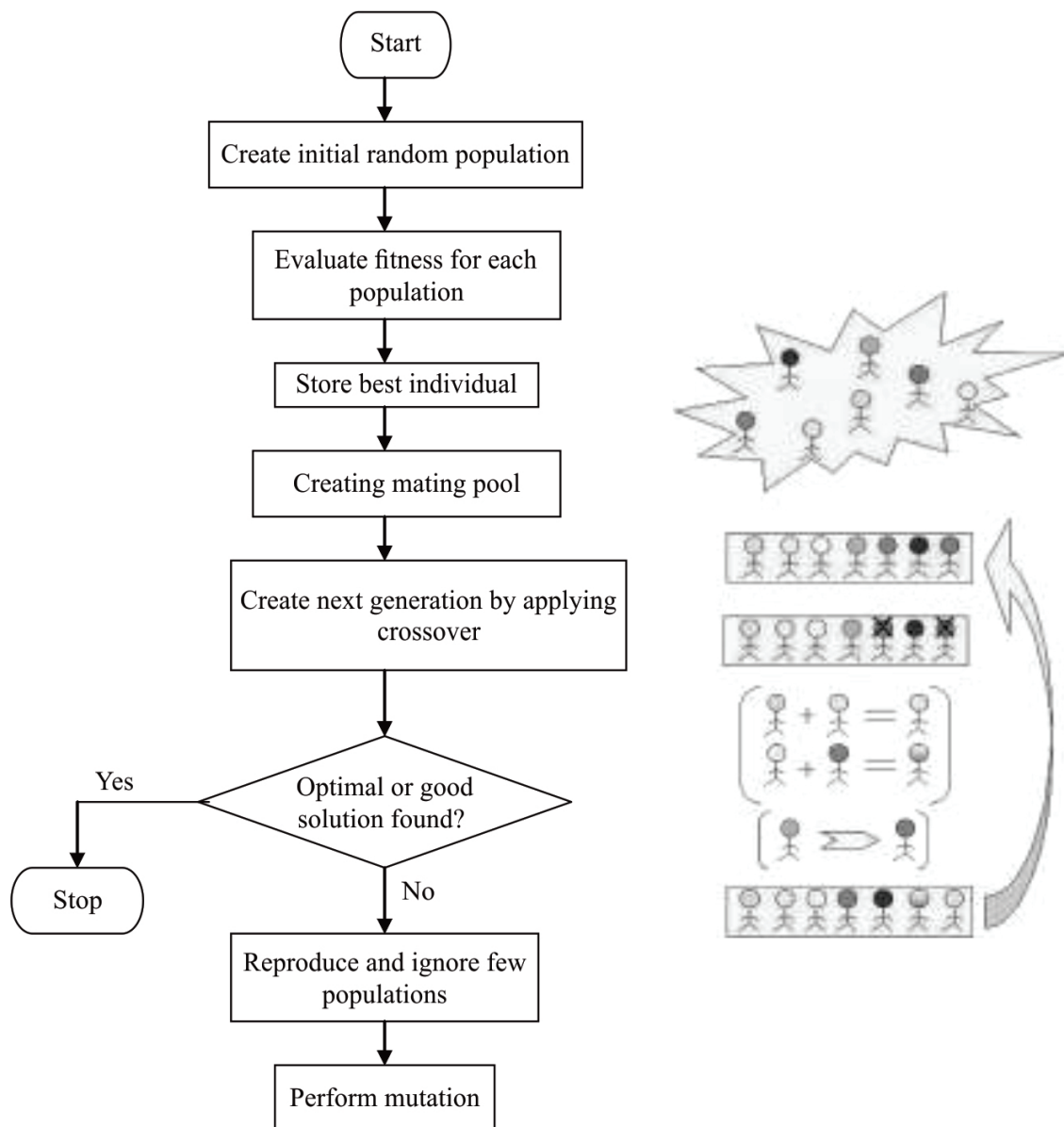
12 GA merupakan metode penyelesaian masalah yang menggunakan genetika sebagai pemodelannya.
 13 Suatu calon solusi dalam GA dimodelkan sebagai suatu individu. Kumpulan individu-individu
 14 ini disebut dengan populasi. Setiap individu dalam populasi direpresentasikan dengan kromosom.
 15 Kromosom merupakan kumpulan parameter yang membentuk suatu solusi. Parameter-parameter
 16 yang menyusun kromosom disebut dengan gen. Setiap kromosom memiliki suatu nilai yang terkait
 17 dengan *fitness* dari solusi yang direpresentasikannya. Nilai *fitness* masing-masing individu akan
 18 digunakan dalam proses seleksi. Individu yang lebih "kuat" akan memiliki peluang terpilih lebih
 19 besar dalam proses seleksi. Individu yang terpilih pada proses seleksi akan disilangkan untuk
 20 menghasilkan keturunan. Keturunan yang dihasilkan pada proses persilangan dapat memiliki
 21 peluang untuk mengalami mutasi. Setelah itu, akan dilakukan proses evaluasi terhadap nilai *fitness*
 22 dari setiap kromosom untuk menentukan apakah solusi yang baik sudah ditemukan atau belum.
 23 Proses ini akan diulang terus-menerus untuk setiap generasi sampai ditemukan solusi yang sudah
 24 optimal. Secara umum, *flowchart* dari GA ditunjukkan pada Gambar 2.4.

2.3.1 *Fitness*

26 Fitness dari suatu individu dalam algoritma genetika adalah suatu nilai fungsi objektif untuk
 27 fenotipenya[4]. *Fitness* harus bisa memperkirakan seberapa dekat sebuah calon solusi dengan solusi
 28 yang optimal. Suatu solusi yang optimal akan memaksimalkan fungsi *fitness*.

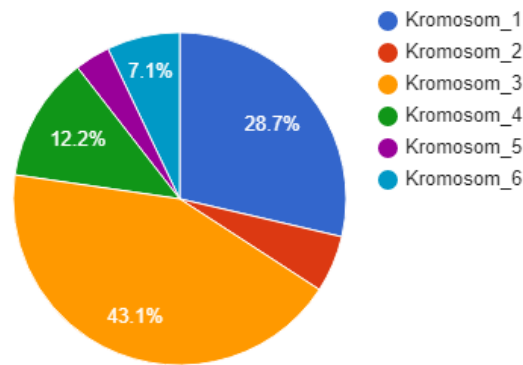
2.3.2 Seleksi

30 Seleksi adalah proses pemilihan dua induk dari populasi untuk disilangkan [4]. Tujuan dari proses
 31 seleksi adalah untuk menonjolkan individu yang memiliki nilai *fitness* tinggi dalam populasi dengan
 32 harapan keturunannya akan memiliki *fitness* yang lebih tinggi. Seleksi adalah suatu metode yang
 33 memilih kromosom secara acak dari populasi berdasarkan fungsi *fitness*. Semakin tinggi fungsi
 34 *fitness* maka semakin tinggi peluang suatu individu akan terpilih.

Gambar 2.4: *flowchart* dari algoritma genetika.

Salah satu teknik yang populer digunakan dalam seleksi adalah *roulette-wheel selection* atau *fitness proportional selection*. *Roulette-wheel selection* memilih suatu individu dari populasi dengan probabilitas yang sebanding dengan nilai *fitness* relatifnya. Hal ini serupa dengan sebuah diagram lingkaran pada Gambar 2.5 di mana setiap kromosom telah dialokasikan sebuah bagian pada diagram sesuai dengan nilai *fitness* relatif. Semakin tinggi nilai *fitness*, maka semakin besar bagian yang dialokasikan dan semakin besar kemungkinan kromosom tersebut akan terpilih dalam proses seleksi.

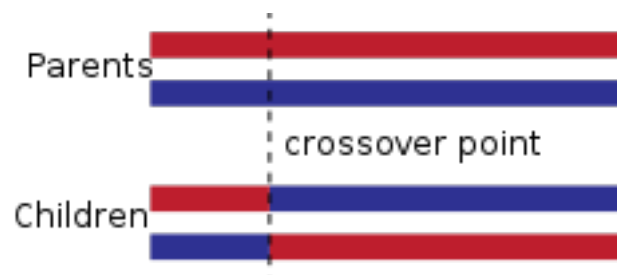
Proses seleksi dalam GA akan memilih sejumlah induk yang cukup untuk reproduksi dan membentuk generasi selanjutnya. Untuk meningkatkan performa GA, dapat juga diterapkan diterapkan strategi *elitism* yaitu dengan langsung memindahkan satu atau beberapa individu dengan nilai *fitness* terbaik ke generasi selanjutnya. Sisanya akan dilakukan dengan cara yang sudah ditentukan sebelumnya seperti *roulette-wheel selection*. Hal ini dilakukan untuk mencegah individu tersebut terhilang dari populasi dalam proses reproduksi.



Gambar 2.5: Diagram Lingkaran

2.3.3 Persilangan

Persilangan adalah operasi genetik yang digunakan untuk menggabungkan informasi genetik dari dua induk untuk menghasilkan keturunan baru [4]. Persilangan dilakukan untuk menghasilkan suatu individu baru yang diharapkan memiliki *fitness* yang lebih baik daripada orangtuanya. Salah satu teknik yang dapat digunakan dalam persilangan adalah *Single-point crossover*. Sebuah titik pada kedua induk dipilih untuk menjadi titik persilangan (*crossover point*). Bit yang berada di sebelah kanan titik persilangan bertukar antara kedua kromosom induk seperti yang ditunjukkan pada Gambar 2.6.

Gambar 2.6: *Single-point crossover*

2.3.4 Mutasi

Mutasi adalah suatu operator genetik yang digunakan untuk mempertahankan keragaman genetik dari satu generasi populasi dalam algoritma genetika. Selain itu mutasi juga dapat mencegah GA terjebak di *local optimum* [4]. Mutasi mengubah satu atau beberapa nilai dalam gen. Mutasi terjadi berdasarkan probabilitas mutasi yang sudah ditentukan sebelumnya. Probabilitas mutasi menentukan seberapa sering kromosom akan dimutasi. Jika tidak terjadi mutasi, maka keturunannya akan langsung masuk ke populasi setelah persilangan tanpa perubahan. Apabila terjadi mutasi, satu atau beberapa bagian dari kromosom akan diubah. Probabilitas ini seharusnya tidak dilakukan terlalu sering, karena jika terlalu sering dilakukan maka GA akan menjadi sama dengan algoritma pencarian acak primitif (*primitive random search*). Mutasi akan dilakukan dengan mengubah nilai dari suatu gen yang telah dipilih menjadi nilai lainnya dengan teknik tertentu (tergantung struktur data dari gen yang akan diubah).

2.3.5 GA dalam Pengelompokan

Meskipun pada umumnya algoritma *K-means* digunakan dalam pengelompokan, tetapi ternyata algoritma *K-means* masih memiliki kekurangan yaitu masih dapat terjebak pada *local optimum*. Oleh karena itu, pada penelitian ini digunakan algoritma genetika sebagai solusi dari permasalahan *local optimum*. Algoritma genetika memiliki kemampuan untuk menghindari terjadinya *local optimum* karena algoritma genetika merupakan algoritma pencarian yang bersifat stokastik dengan memperbolehkan terjadinya variasi acak dalam proses pencarian solusinya. Oleh karena itu, diharapkan pengelompokan berbasis GA dapat menghasilkan solusi yang lebih baik dibandingkan dengan pengelompokan pada umumnya yang menggunakan algoritma *K-means*.

Algorithm 2 Algoritma Genetika[8]

function Algoritma-Genetika(*populasi*) **returns** solusi berupa individu

inputs: *populasi*, himpunan individu

```

1: solusi  $\leftarrow$  himpunan solusi tiap generasi
2: repeat
3:   tambahkan individu dengan nilai fitness tertinggi dari populasi ke solusi
4:   populasi_baru  $\leftarrow$  himpunan kosong
5:   for  $i=1$  to Size(populasi) do
6:      $x \leftarrow$  Seleksi-acak(populasi)
7:      $y \leftarrow$  Seleksi-acak(populasi)
8:     anak  $\leftarrow$  Persilangan( $x, y$ )
9:     rand  $\leftarrow$  Random(0,1)
10:    if rand  $\leq$  prob_mutasi then
11:      anak  $\leftarrow$  Mutasi(anak)
12:    end if
13:    tambahkan anak ke populasi_baru
14:  end for
15:  populasi  $\leftarrow$  populasi_baru
16: until  $N$  solusi terakhir pada solusi tidak memiliki perubahan yang signifikan
17: return individu terbaik dalam populasi, berdasarkan nilai fitness

```

Penjelasan untuk fungsi Algoritma-Genetika pada Algoritma 2 adalah sebagai berikut:

- Pada baris 1, variabel *solusi* berfungsi untuk mencatat sejarah dari solusi yang pernah dihasilkan pada setiap generasi.
- Pada baris 3, dilakukan pencatatan solusi pada generasi saat ini ke variabel *solusi*.
- Pada baris 9, pembangkitan bilangan riil secara acak antara 0 sampai dengan 1 dengan distribusi *uniform*.
- Pada baris 16, nilai *fitness* pada N solusi terakhir sudah mengalami konvergensi.
- Pada baris 17, mengembalikan individu terbaik dalam populasi terakhir GA.

Penjelasan untuk fungsi Persilangan pada Algoritma 2 adalah sebagai berikut:

- Pada baris 2, akan dibangkitkan suatu bilangan bulat antara 1 sampai N dengan distribusi *uniform*
- Pada baris 3, *Append* merupakan fungsi untuk menggabungkan dua *string* dan *Substring* merupakan fungsi untuk memotong *string* mulai dari batas tertentu sampai dengan batas tertentu juga.

function Persilangan(x,y) **returns** anak berupa individu

inputs: x dan y , individu induk

```

1:  $n \leftarrow \text{Length}(x)$ 
2:  $c \leftarrow \text{Random}(1,N)$ 
3: return Append(Substring( $x,1,c$ ), Substring( $y,c+1,n$ ))

```

function Seleksi-acak($populasi$) **returns** sebuah individu hasil seleksi

inputs: $populasi$, populasi saat ini

```

1:  $sum \leftarrow 0$ 
2: for all  $individu \in populasi$  do
3:    $sum \leftarrow sum + \text{Fitness}(individu)$ 
4: end for
5:  $terpilih \leftarrow \text{Random}(0,1) \times sum$ 
6: for all  $individu \in populasi$  do
7:    $terpilih \leftarrow terpilih - \text{Fitness}(individu)$ 
8:   if  $terpilih \leq 0$  then
9:     return  $individu$ 
10:  end if
11: end for
12: return individu dengan urutan terakhir di  $populasi$ 

```

- 1 Penjelasan untuk fungsi Seleksi-acak pada Algoritma 2 adalah sebagai berikut:
- 2 • Pada baris 3, fungsi *Fitness* akan mengembalikan nilai *fitness* dari individu yang menjadi
- 3 parameternya.
- 4 • Pada baris 5, akan dibangkitkan suatu bilangan riil secara acak antara 0 sampai sum dengan
- 5 cara mengalikan bilangan riil acak antara 0 sampai 1 dan sum .

function Mutasi($individu$) **returns** individu hasil mutasi

inputs: $individu$, individu yang akan dilakukan mutasi

```

1:  $n \leftarrow \text{Length}(x)$ 
2:  $c \leftarrow \text{Random}(1,n)$ 
3: ubah nilai gen ke- $c$  pada  $individu$  {nilai bervariasi tergantung metode}
4: return  $individu$ 

```

- 6 Penjelasan untuk fungsi Mutasi pada Algoritma 2 adalah sebagai berikut:
- 7 • Pada baris 1, fungsi *Length* akan mengembalikan panjang dari *string* parameternya.
- 8 • Pada baris 2, fungsi *Random* akan mengembalikan bilangan bulat acak antara 1 sampai N .

2.4 Model Ruang Vektor

Model ruang vektor adalah representasi dari koleksi dokumen sebagai vektor dalam ruang vektor yang umum [9]. Model ruang vektor ini biasanya digunakan dalam sejumlah operasi pencarian informasi mulai dari penilaian dokumen pada *query*, klasifikasi dokumen, dan pengelompokan dokumen. Pada penerapannya, akan dilakukan pengukuran kemiripan suatu dokumen terhadap *query* untuk dapat menentukan peringkat relevansi dokumen terhadap *query* (*relevance ranking*). Dokumen dan *query* akan direpresentasikan sebagai model ruang vektor seperti pada Persamaan 2.2.

$$d_i = (w_{1,i}, w_{2,i}, \dots, w_{n,i}) \quad (2.2)$$

dengan d_i merupakan dokumen ke- i , $w_{n,i}$ merupakan bobot dari *term* n untuk dokumen i . Setiap dimensi pada vektor tersebut menggambarkan *term* berbeda dalam dokumen. Kemiripan antara dokumen dan *query* akan ditentukan dengan mengukur perbedaan sudut antara vektor dokumen dan vektor *query*. Semakin kecil sudut antara dokumen dan *query*, maka dokumen dan *query* dianggap semakin mirip. Namun pada praktiknya, dilakukan perhitungan jarak cosinus untuk menggantikan pengukuran sudut antara dua vektor karena jarak cosinus berbanding terbalik dengan besar sudut antara dua vektor sehingga tidak perlu dilakukan perhitungan lebih lanjut untuk mendapatkan besar sudutnya. Jarak cosinus dapat dihitung menggunakan Persamaan 2.3.

$$s_{ij} = \frac{i \cdot j}{\|i\| \times \|j\|} \quad (2.3)$$

dengan s_{ij} adalah kesamaan antara vektor ke- i dengan vektor ke- j , i adalah vektor ke- i , dan j adalah vektor ke- j . Persamaan ini menjelaskan bahwa semakin kecil sudut antara dua vektor, maka tingkat kemiripannya semakin besar.

2.5 Pembobotan *Term* (*Term Weighting*)

Term merupakan suatu kelas dari kumpulan karakter pada suatu dokumen tertentu yang digabungkan sebagai unit semantik dan tercatat dalam kamus (*dictionary*) suatu sistem [9]. Pembobotan *term* merupakan suatu proses menentukan nilai dari suatu *term* dalam sebuah dokumen. Pembobotan *term* bertugas untuk memetakan *term* kepada suatu nilai numerik yang merepresentasikan seberapa penting *term* tersebut dalam suatu dokumen. Tidak semua *term* dalam dokumen itu penting sehingga dengan memberikan nilai kepada masing-masing *term* dapat dengan tepat merepresentasikan isi dokumen. Secara umum, apabila suatu *term* semakin penting dalam suatu dokumen (semakin menggambarkan isi dokumen), maka nilai bobotnya akan semakin besar. Sebaliknya jika suatu *term* semakin tidak penting (kata-kata yang umum digunakan seperti kata sambung, kata ganti, dan lain-lain), maka nilai bobotnya akan semakin kecil.

Ada banyak cara untuk menghitung bobot suatu *term*, dua diantaranya adalah bobot frekuensi (*Frequency weighting*) dan Bobot TF-IDF (*TF-IDF weighting*).

2.5.1 Bobot frekuensi

Bobot frekuensi merupakan teknik pembobotan yang sangat sederhana karena bobotnya merupakan jumlah kemunculan *term* tersebut dalam dokumen. Bobot frekuensi dapat digambarkan dengan Persamaan 2.4

$$w_i = tf_i \quad (2.4)$$

dengan w_i merupakan bobot *term* ke- i dan tf_i merupakan frekuensi kemunculan *term* ke- i pada dokumen. Akan digunakan empat buah dokumen (masing-masing terdiri dari satu kalimat) yang berasal dari contoh pada [9] sebagai berikut:

- 1 **Doc 1** new home sales top forecasts
- 2 **Doc 2** home sales rise in july
- 3 **Doc 3** increase in home sales in july
- 4 **Doc 4** july new home sales rise

5 Berdasarkan keempat contoh dokumen, maka dibentuk tabel ketetanggaan antara *term* dengan
6 dokumen pada Tabel 2.1.

<i>Term</i>	doc 1	doc 2	doc 3	doc 4
new	1	0	0	1
home	1	1	1	1
sales	1	1	1	1
top	1	0	0	0
forecast	1	0	0	0
rise	0	1	0	1
in	0	1	2	0
july	0	1	1	1
increase	0	0	1	0

Tabel 2.1: *Term-document incidence matrix*

7 *Term* "in" dalam tabel pada *doc* 1 dan *doc* 4 bernilai 0 karena *term* "in" tidak muncul pada
8 *doc* 1 dan *doc* 4. Sedangkan pada *doc* 2, *term* "in" muncul satu kali sehingga bernilai 1 pada tabel.
9 Begitu juga pada *doc* 3, *term* "in" muncul dua kali sehingga bernilai 2 pada tabel.

10 2.5.2 Bobot TF-IDF

11 Pada bobot frekuensi, bobot hanya dihitung berdasarkan kemunculan *term* dalam dokumen itu
12 sendiri. Namun dalam bobot TF-IDF (*term frequency-inverse document frequency*), bobot juga
13 dihitung berdasarkan kemunculan *term* pada himpunan dokumen. Metode ini sangat populer
14 digunakan oleh sistem rekomendasi berbasis teks [10]. Rumus dari pembobotan menggunakan
15 TF-IDF ditunjukkan oleh Persamaan 2.5.

$$w_{i,d} = tf_{i,d} \times \log \frac{N}{N_i} \quad (2.5)$$

$$tf_{i,d} = \frac{f_{i,d}}{\sum_{t \in d} f_{t,d}}$$

16 dengan $w_{i,d}$ merupakan bobot *term* i dalam dokumen d , $tf_{i,d}$ merupakan frekuensi *term* i pada
17 dokumen d , N menyatakan banyaknya anggota himpunan dokumen, N_i menunjukkan frekuensi
18 dokumen dari *term* i (banyaknya dokumen pada himpunan dokumen yang memuat *term* i), $f_{i,d}$
19 merupakan banyaknya kemunculan *term* i dalam dokumen d , $t \in d$ merupakan seluruh *term* t
20 yang merupakan anggota dari dokumen d , dan $f_{t,d}$ merupakan banyaknya kemunculan *term* t pada
21 dokumen d .

22 Berdasarkan rumus tersebut, maka dapat ditarik dua kesimpulan yaitu:

- 23 • Semakin sering suatu *term* muncul di suatu dokumen, maka semakin representatif *term*
24 tersebut terhadap isi dokumen.
- 25 • Semakin banyak dokumen yang memuat suatu *term*, maka nilai informasi *term* tersebut
26 semakin kecil. Hal ini terjadi karena *term* yang muncul di mayoritas dokumen tidak dapat
27 menggambarkan arti dari suatu dokumen.

Metode penetapan bobot TF-IDF dianggap sebagai metode yang berkinerja baik karena mempertimbangkan frekuensi kemunculan *term* baik secara lokal (TF) maupun global (IDF).

2.6 Metrik *Intraccluster* untuk Mengukur Kinerja Metode *Clustering*

Untuk mengukur performa dari suatu algoritma, perlu dilakukan pengujian. Pengujian ini dapat menentukan apakah algoritma yang digunakan sudah cukup baik dalam menyelesaikan masalah yang dibuat. Dalam pengelompokan, ada beberapa cara untuk mengukur apakah objek-objek sudah berhasil dikelompokkan secara baik atau tidak. Cara yang pertama adalah dengan mengukur jarak antara tiap objek ke titik pusat *cluster* (*centroid*) atau biasa disebut jarak *intraccluster*. Lalu cara yang kedua adalah mengukur jarak antar kelompok yang dapat diukur dengan cara menghitung jarak setiap *centroid* ke *centroid* lainnya.

Metrik yang digunakan [2] dalam mengukur kinerja suatu metode pengelompokan adalah perhitungan jarak *intraccluster*. Jarak *intraccluster* dapat diukur dengan cara menjumlahkan jarak setiap objek ke masing-masing titik pusatnya. Cara untuk menghitung jarak *intraccluster* ditunjukkan dalam Persamaan 2.6.

$$M = \sum_{i=1}^K M_i, \quad (2.6)$$

$$M_i = \sum_{x_j \in C_i} \|x_j - z_i\|$$

dengan M merupakan jumlah jarak seluruh objek ke *centroid* masing-masing, K merupakan banyaknya *cluster*, M_i merupakan jumlah jarak seluruh objek anggota *cluster* ke- i ke titik pusatnya, x_j merupakan objek ke- j , C_i merupakan *centroid* ke- i , dan z_i merupakan *centroid* dari *cluster* ke- i . Semakin kecil nilai M , maka pengelompokan dianggap semakin baik karena jarak setiap objek ke titik pusatnya dekat. Metrik ini dapat mengukur seberapa baik objek sudah dikelompokkan dengan mempertimbangkan kedekatan setiap objek ke titik pusatnya.

BAB 3

ANALISIS

Bab ini membahas hasil analisis berdasarkan dasar teori yang sudah dijelaskan sebelumnya. Pada bab ini akan dijelaskan hasil analisis *dataset* yang akan digunakan dalam pengujian, representasi dokumen dalam perangkat lunak, dan pemodelan ruang vektor pada dokumen. Selain itu pada bab ini juga akan dibahas mengenai representasi kromosom, fungsi *fitness*, dan beberapa operasi genetik yang akan digunakan dalam membuat pengelompokan dokumen berbasis algoritma genetika.

3.1 Analisis *Dataset*

Pada bagian ini akan dibahas mengenai *dataset* yang akan digunakan dalam proses pengujian. *Dataset* yang akan digunakan berisi artikel berita *BBC News* dan disediakan untuk menjadi tolok ukur dalam penelitian pembelajaran mesin. Karakteristik dari *dataset* ini antara lain:

- Terdiri dari 2225 dokumen yang berasal dari *website BBC News* dari tahun 2004-2005.
- Dokumen ditulis dalam Bahasa Inggris.
- Terbagi menjadi lima topik yaitu *business*, *entertainment*, *politics*, *sport*, dan *tech*.
- Pada topik *business* terdapat 510 dokumen, *entertainment* terdapat 386 dokumen, *politics* terdapat 417 dokumen, *sport* terdapat 511 dokumen, dan *tech* terdapat 401 dokumen.
- Dokumen merupakan *plain text* yang ditulis dalam file dengan ekstensi *TXT*.
- Rata-rata dalam satu dokumen terdapat 384 kata.

3.2 Representasi Dokumen

Dokumen tidak bisa langsung digunakan begitu saja dalam proses pengelompokan. Tidak seperti manusia yang dapat melakukan proses secara manual, komputer tidak dapat menemukan nilai informasi dari data mentah berupa dokumen. Dokumen yang ada perlu direpresentasikan menjadi bentuk yang bisa diambil informasinya baru kemudian dapat diolah dalam proses lebih lanjut. Model ruang vektor (Subbab 2.4) digunakan dalam penelitian ini untuk merepresentasikan dokumen sehingga informasinya dapat diproses dengan lebih mudah.

3.3 Model Ruang Vektor

Pada subbab 2.4 telah dijelaskan bahwa dokumen akan dibentuk ke dalam sebuah vektor yang memiliki banyak dimensi berdasarkan banyaknya *term* berbeda dalam dokumen. Seperti yang telah dibahas pada subbab 2.5, ada dua cara untuk menentukan bobot dari suatu *term* dalam dokumen yaitu bobot frekuensi dan bobot tf-idf. Sebagai contoh akan digunakan tiga dokumen berikut:

1. Penjualan properti di Indonesia meningkat di Bulan Februari.

2. Penjualan asuransi kendaraan di Indonesia meningkat.

3. Bulan Februari merupakan bulan puncak penjualan kendaraan.

Berdasarkan tiga dokumen tersebut akan ditentukan bobot dari masing-masing *term* dalam tiap dokumen untuk setiap jenis pembobotan.

3.3.1 Bobot Frekuensi

Pada subbab 2.5.1 telah dijelaskan bahwa bobot frekuensi dari suatu *term* dapat ditentukan dengan cara menghitung banyaknya kemunculan *term* tersebut dalam dokumen. Hasil perhitungan bobot frekuensi berdasarkan contoh pada subbab 3.3 ditunjukkan dalam tabel 3.1.

<i>Term</i>	Bobot		
	Dokumen 1	Dokumen 2	Dokumen 3
penjualan	1	1	1
properti	1	0	0
di	2	1	0
indonesia	1	1	0
meningkat	1	1	0
bulan	1	0	2
februari	1	0	1
asuransi	0	1	0
kendaraan	0	1	1
merupakan	0	0	1
puncak	0	0	1

Tabel 3.1: Hasil perhitungan bobot frekuensi

Semakin besar bobot, maka *term* dianggap semakin mewakili isi dokumen. Sebagai contoh pada tabel 3.1, *term* "di" muncul dua kali pada dokumen 1 sehingga berbobot 2, muncul satu kali pada dokumen 2 sehingga berbobot 1, dan tidak muncul sama sekali pada dokumen 3 sehingga memiliki bobot 0 sehingga *term* "di" dianggap mewakili dokumen 1 karena muncul dua kali pada dokumen tersebut.

3.3.2 Bobot TF-IDF

Berbeda dengan bobot frekuensi yang hanya menghitung frekuensi kemunculan *term* pada dokumen, perhitungan bobot TF-IDF ini memerlukan perhitungan seperti yang telah dijelaskan dalam persamaan 2.5 pada subbab 2.5.2. Hasil perhitungan bobot TF-IDF berdasarkan contoh pada Subbab 3.3 ditunjukkan dalam Tabel 3.2.

Sama halnya dengan bobot frekuensi, semakin besar bobotnya maka *term* tersebut semakin mewakili isi dokumen. *Term* "penjualan" muncul di ketiga dokumen sehingga pada saat perhitungan IDF menghasilkan nilai nol ($\log \frac{N}{N_i} = \log \frac{3}{3} = 0$). Dapat disimpulkan bahwa *term* "penjualan" tidak mewakili dokumen manapun karena muncul di semua dokumen. Untuk menjelaskan nilai dalam tabel, dijelaskan salah satu contoh perhitungan TF-IDF untuk *term* "properti" pada dokumen 1 ditunjukkan dalam Persamaan 3.1 dan 3.2.

$$tf_{i,d} = \frac{f_{i,d}}{\sum_{t \in d} f_{t,d}} \quad (3.1)$$

$$tf_{\text{"properti"},1} = \frac{1}{8} = 0.125$$

<i>Term</i>	Bobot		
	Dokumen 1	Dokumen 2	Dokumen 3
penjualan	0	0	0
properti	0.0596	0	0
di	0.0440	0.0293	0
indonesia	0.0220	0.0293	0
meningkat	0.0220	0.0293	0
bulan	0.0220	0	0.0503
februari	0.0220	0	0.0252
asuransi	0	0.0795	0
kendaraan	0	0.0293	0.0252
merupakan	0	0	0.0682
puncak	0	0	0.0682

Tabel 3.2: Hasil perhitungan bobot TF-IDF

$$w_{i,d} = tf_{i,d} \times \log \frac{N}{N_i} \quad (3.2)$$

$$w_{\text{"properti"},1} = 0.125 \times \log \frac{3}{1} = 0.125 \times 0.4771 = 0.0596$$

3.4 Representasi Kromosom

Setiap *string* kromosom merupakan deretan bilangan riil yang merepresentasikan K titik pusat *cluster* (*centroid*) dalam bentuk vektor. Dalam ruang N dimensi, panjang dari kromosom akan menjadi $N \times K$ gen. N kata pertama merepresentasikan N dimensi dari *centroid* pertama, N kata selanjutnya merepresentasikan N dimensi dari *centroid* kedua, dan seterusnya. Sebagai contoh apabila diketahui ada tiga buah *centroid* dalam bidang dua dimensi **C1** (3.05, 1.43), **C2** (15.85, 14.23), dan **C3** (5.12, 9.45). Hasil representasi ketiga *centroid* pada kromosom ditunjukkan oleh Tabel 3.3.

C1		C2		C3	
3.05	1.43	15.85	14.23	5.12	9.45

Tabel 3.3: Representasi *centroid* ke dalam kromosom

3.5 Fungsi *Fitness*

Perhitungan *fitness* dalam penelitian ini terdiri dari dua tahap. Pada tahap pertama, terjadi pembentukan *cluster* berdasarkan titik pusat yang terkandung dalam kromosom. Hal ini dilakukan dengan menetapkan setiap titik $x_i, i = 1, 2, \dots, n$ ke dalam sebuah *cluster* C_j dengan *centroid* z_j sehingga

$$\|x_i - z_j\| < \|x_i - z_p\|, p = 1, 2, \dots, K, \text{ dan } p \neq j. \quad (3.3)$$

Setelah proses pengelompokan selesai, titik pusat yang terkandung dalam kromosom diganti dengan rata-rata titik dari tiap *cluster*. Dengan kata lain, untuk *cluster* C_i , *centroid* baru z_i^* dapat

1 dihitung menggunakan Persamaan 3.4

$$z_i^* = \frac{1}{n_i} \sum_{x_j \in C_i} x_j, i = 1, 2, \dots, K. \quad (3.4)$$

2 dengan z_i^* merupakan titik pusat *cluster* ke- i , n_i merupakan jumlah anggota *cluster* ke- i , dan
 3 x_j merupakan titik ke- j yang merupakan anggota dari *cluster* ke- i . z_i^* ini akan menggantikan z_i
 4 sebelumnya di kromosom. Ada dua metode perhitungan fungsi fitness yang diimplementasikan
 5 dalam penelitian ini yaitu *cosine similarity*.

6 3.5.1 Cosine Similarity

Perhitungan *fitness* menggunakan *cosine similarity* dapat dilakukan dengan Persamaan 3.5

$$f = \sum_{i=1}^K f_i, \quad (3.5)$$

$$f_i = \sum_{x_j \in C_i} \frac{x_j \cdot z_i}{\|x_j\| \times \|z_i\|}$$

7 semakin besar nilai dari fungsi *fitness* f , maka kromosom tersebut semakin mendekati solusi
 8 yang optimal.

9 3.6 Operasi Genetik

10 Ada beberapa operasi genetik yang akan dibahas, di antaranya: inisialisasi populasi, seleksi,
 11 persilangan, dan mutasi.

12 3.6.1 Inisialisasi Populasi

13 K *centroid* yang terkandung dalam kromosom pada mulanya dipilih secara acak sebanyak K titik
 14 dari keseluruhan himpunan data. Lalu proses ini diulang sebanyak P kali di mana P merupakan
 15 ukuran populasi yang diinginkan.

16 3.6.2 Seleksi

17 Proses seleksi ini terjadi berdasarkan konsep *survival of the fittest* yang diadaptasi dari sistem
 18 genetika alami. Konsep ini mengatakan bahwa hanya individu terbaik yang akan bertahan hidup
 19 (lolos seleksi alam). Dengan mengadaptasi konsep tersebut, GA menerapkan seleksi berdasarkan
 20 nilai *fitness* tiap individu. Dalam penelitian ini, calon induk dari generasi selanjutnya dipilih dengan
 21 menggunakan teknik *roulette-wheel selection*. Seperti yang telah dijelaskan dalam Subbab 2.3.2,
 22 dalam teknik *roulette-wheel selection* individu yang memiliki nilai *fitness* lebih tinggi akan memiliki
 23 peluang lebih besar untuk terpilih dan menjadi induk bagi generasi berikutnya. Namun karena
 24 masih ada peluang individu dengan nilai *fitness* terbaik tidak terpilih, maka dalam penelitian ini
 25 juga akan digunakan teknik elitisme [11]. Dengan digunakannya elitisme, maka individu dengan
 26 nilai *fitness* terbaik akan disalin dan langsung menjadi anggota dari generasi berikutnya. Hal ini
 27 akan menjamin individu terbaik tidak akan hilang akibat tidak terpilih oleh *roulette-wheel selection*.

28 3.6.3 Persilangan

29 Persilangan dalam penelitian ini terjadi terhadap dua induk dengan satu titik potong (*single-point*
 30 *crossover*). Misalkan kromosom memiliki panjang l , sebuah angka acak akan diambil sebagai titik
 31 potong dalam batas $[1, l - 1]$. Bagian kromosom sebelah kanan titik potong akan ditukar antara
 32 kedua induk sehingga menghasilkan dua individu keturunan.

3.6.4 Mutasi

Setiap kromosom mengalami mutasi dengan probabilitas mutasi tetap μ_c . Apabila mutasi terjadi, maka akan ditentukan gen mana yang akan mengalami mutasi dengan mengambilnya secara acak. Nilai gen yang baru akan ditentukan dari pembangkitan suatu angka acak yang berada antara batas minimum kemunculan istilah (0) dan total kemunculan istilah tersebut dari keseluruhan dokumen. Sebagai contoh dengan menggunakan Tabel 3.3, akan ditentukan satu dari enam gen yang akan mengalami mutasi. Misalkan dalam contoh ini gen kedua yang mengalami mutasi. Lalu akan dilakukan pembangkitan angka acak antara 0 sampai dengan total kemunculan istilah dari keseluruhan dokumen (dalam contoh ini bernilai 9). Kromosom hasil mutasi ditunjukkan dalam Tabel 3.4

C1	C2	C3
3.05	4.18	15.85
14.23	5.12	9.45

Tabel 3.4: Kromosom hasil mutasi

BAB 4

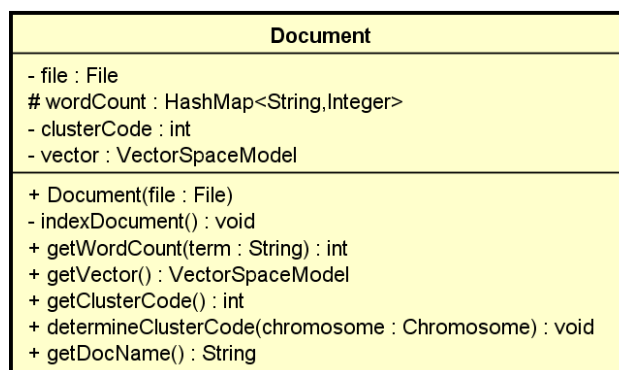
PERANCANGAN

Pada bab ini dijelaskan mengenai beberapa perancangan yang dilakukan dalam penelitian ini yaitu rancangan kelas dan rancangan antarmuka pengguna

4.1 Rancangan Kelas

Berdasarkan hasil analisis dari masalah yang dihadapi, dibentuklah diagram kelas pada gambar 4.11 sebagai gambaran dari perangkat lunak yang akan dibuat.

4.1.1 *Document*



Gambar 4.1: Kelas *Document*

Kelas ini merupakan representasi dari dokumen yang akan diproses dalam pengelompokan. Kelas ini berfungsi untuk menyimpan informasi yang dibutuhkan dari sebuah dokumen selama proses pengelompokan. Atribut yang dimiliki oleh kelas *Document* adalah:

- *wordCount*: bertipe *HashMap* dengan *key* bertipe *String* dan *value* bertipe *Integer*. Atribut ini menyimpan pasangan kata yang dimiliki oleh dokumen tersebut dan frekuensinya.
- *file*: atribut ini bertipe *File* milik *package java.io* yang berfungsi untuk merepresentasikan *file* dari dokumen yang akan diproses.
- *vector*: atribut bertipe *VectorSpaceModel* ini merepresentasikan model ruang vektor pada sebuah dokumen.

Method yang terdapat dalam kelas ini adalah:

- *Document*: merupakan *constructor* dengan sebuah parameter bertipe *File* yaitu *file* dari dokumen yang akan dikelompokkan.

- *indexDocument*: method tanpa kembalian (*void*) yang berfungsi untuk mengindeks dokumen untuk mengisi atribut *wordCount*.
- *getWordCount*: berfungsi untuk mengembalikan banyaknya istilah *term* muncul dalam dokumen.
- *getVector*: merupakan *getter* dari variabel *vector*.
- *determineClusterCode*: berfungsi untuk menentukan *cluster* dari dokumen.
- *getDocName*: mengembalikan nama *file* dari dokumen.

4.1.2 VectorSpaceModel

VectorSpaceModel
termsWeight : HashMap<String, Double> - distanceCalculator : DistanceCalculator
VectorSpaceModel(wordCount : HashMap<String, Integer>, distanceMode : String) # generateVector(wordCount : HashMap<String, Integer>) : void + getWeight(term : String) : double + calculateDistance(otherVSM : VectorSpaceModel) : double + mutate() : void + setTermsWeight(termsWeight : HashMap<String, Double>) : void

Gambar 4.2: Kelas *VectorSpaceModel*

Kelas ini merupakan kelas abstrak yang merepresentasikan model ruang vektor. kelas ini memiliki fungsi-fungsi yang umum dimiliki oleh sebuah model ruang vektor. Atribut yang dimiliki kelas ini adalah:

- *termsWeight*: bertipe *Hashmap* dengan *key* berupa *String* dan *value* berupa *Double*. atribut ini menyimpan pasangan istilah dan bobotnya sesuai dengan metode pembobotan.
- *distanceCalculator*: bertipe *DistanceCalculator* dan merupakan objek yang akan digunakan untuk menghitung jarak antar vektor.

Method yang terdapat dalam kelas ini adalah:

- *VectorSpaceModel*: merupakan constructor dengan dua parameter yaitu *wordCount* bertipe *Hashmap<String, Integer>* yang merupakan pasangan kata dan banyak kemunculannya dalam dokumen serta *distanceMode* bertipe *String* yang akan menentukan tipe perhitungan jarak antar vektor.
- *generateVector*: merupakan *method* tanpa parameter untuk mengubah banyak kemunculan kata menjadi berat.
- *getWeight*: berfungsi untuk mengembalikan berat dari istilah *term*.
- *calculateDistance*: berfungsi untuk menghitung jarak antara vektor ini dengan *otherVSM* menggunakan metode yang dipilih pada parameter di *constructor*.
- *mutate*: merupakan *method* untuk melakukan mutasi pada sebuah dimensi dalam vektor.
- *setTermsWeight*: merupakan setter dari atribut *termsWeight*.

BagOfWordVSM
+ BagOfWordVSM(wordCount : HashMap<String,Integer>, distanceMode : String) + getWeight(term : String) : double # generateVector(wordCount : HashMap<String,Integer>) : void

Gambar 4.3: Kelas *BagOfWordVSM*

4.1.3 *BagOfWordVSM*

Kelas ini merupakan kelas yang mengimplementasikan kelas *VectorSpaceModel*. Kelas ini menggunakan metode pembobotan *BagOfWord* di mana bobot tiap istilah merupakan banyaknya kemunculan istilah itu sendiri. Atribut yang dimiliki kelas ini seluruhnya merupakan atribut yang diturunkan dari kelas *VectorSpaceModel* dan hanya melakukan *override* pada *method* yang masih bersifat abstrak.

4.1.4 *DistanceCalculator*

DistanceCalculator
+ calculateDistance(vsm1 : VectorSpaceModel, vsm2 : VectorSpaceModel) : double

Gambar 4.4: Kelas *DistanceCalculator*

Kelas ini merupakan kelas abstrak yang berfungsi untuk menghitung jarak antara dua buah objek bertipe *VectorSpaceModel*. Kelas ini tidak memiliki atribut dan hanya memiliki sebuah *method* abstrak yaitu *calculateDistance* yang memiliki dua buah parameter *vsm1* dan *vsm2*. Hasil yang dikembalikan oleh *method* ini adalah jarak dari kedua vektor tersebut sesuai dengan metode perhitungan jaraknya.

4.1.5 *EuclideanDistanceCalculator*

EuclidianDistanceCalculator
+ calculateDistance(vsm1 : VectorSpaceModel, vsm2 : VectorSpaceModel) : double

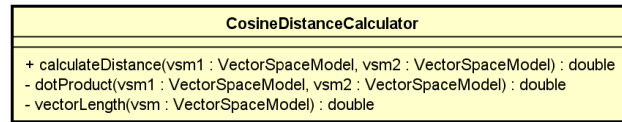
Gambar 4.5: Kelas *EuclideanDistanceCalculator*

Kelas ini mengimplementasikan kelas abstrak *DistanceCalculator*. Kelas ini hanya melakukan *override* pada *method* *calculateDistance* dengan melakukan perhitungan menggunakan jarak euclidean untuk menghitung jarak antara dua buah vektor (Subbab ??).

4.1.6 *CosineDistanceCalculator*

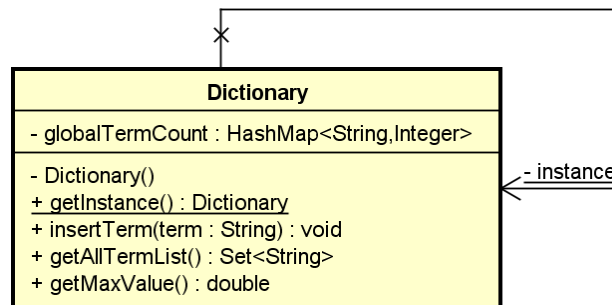
Kelas ini juga mengimplementasikan kelas abstrak *DistanceCalculator*. Kelas ini memiliki dua *method* tambahan selain melakukan *override* pada *method* *calculateDistance*. *Method* yang ada pada kelas ini adalah:

- *calculateDistance*: merupakan *method* yang diturunkan dari kelas *DistanceCalculator*. *Method* ini mengembalikan jarak dari *vsm1* dan *vsm2* yang dihitung menggunakan persamaan cosinus (Subbab ??).
- *dotProduct*: berfungsi untuk menghitung hasil perkalian titik (*dot product*) antara *vsm1* dan *vsm2*.

Gambar 4.6: Kelas *CosineDistanceCalculator*

- *vectorLength*: berfungsi untuk menghitung panjang dari vektor *vsm*.

4.1.7 Dictionary

Gambar 4.7: Kelas *Dictionary*

Kelas ini merepresentasikan sebuah kamus yang menangani seluruh kebutuhan dalam proses pengelompokan yang membutuhkan akses global untuk keseluruhan koleksi dokumen. Atribut yang ada dalam kelas ini adalah:

- *globalTermCount*: bertipe *HashMap* dengan *key* bertipe *String* dan *value* bertipe *Integer*. Atribut ini berfungsi untuk menyimpan seluruh istilah yang muncul dan banyak kemunculannya dalam keseluruhan koleksi dokumen.
- *instance*: merupakan objek bertipe *Dictionary* sebagai instansiasi satu-satunya dari kelas *Dictionary* karena kelas ini bersifat *singleton*.

Method yang ada pada kelas ini adalah:

- *Dictionary*: merupakan *constructor private* untuk menjamin tidak akan ada lebih dari satu *instance* selama perangkat lunak berjalan.
- *getInstance*: merupakan *method static* yang berfungsi sebagai *getter* dari atribut *instance*.
- *insertTerm*: berfungsi untuk memasukkan istilah *term* ke dalam variabel *globalTermCount*.
- *getAllTermList*: bertugas mengembalikan daftar seluruh istilah yang pernah muncul di seluruh koleksi dokumen.
- *getValue*: bertugas mengembalikan banyaknya kata *term* muncul dalam seluruh koleksi dokumen.

4.1.8 Gene

Kelas ini merepresentasikan gen dalam algoritma genetika. Kelas ini hanya memiliki sebuah atribut *value* bertipe *VectorSpaceModel*. Atribut ini menyimpan model ruang vektor yang menjadi titik pusat *cluster* (*centroid*). Method yang ada pada kelas ini adalah:

Gene
- value : VectorSpaceModel
+ Gene(value : VectorSpaceModel) + getValue() : VectorSpaceModel + mutate() : void + setTermsWeight(termsWeight : HashMap<String,Double>) : void

Gambar 4.8: Kelas *Gene*

- *Gene*: merupakan *constructor* dari kelas *Gene* yang membutuhkan sebuah parameter bertipe *VectorSpaceModel* untuk mengisi variabel *value*.
- *getValue*: merupakan *getter* dari atribut *value*.
- *mutate*: berfungsi untuk melakukan mutasi pada gen. *Method* ini sebenarnya hanya bertugas memanggil fungsi *mutate()* dari atribut *value*.
- *setTermsWeight*: berfungsi untuk mengubah nilai atribut *termsWeight* milik atribut *value*.

4.1.9 *Chromosome*

Chromosome
+ <u>MUTATION_PROBABILITY : double = .20</u> - rand : Random - genes : List<Gene>
+ Chromosome() + addGene(gene : Gene) : void + mutate() : void + crossover(otherChromosome : Chromosome) : Chromosome + computeFitness() : double + getAllGenes() : List<Gene>

Gambar 4.9: Kelas *Chromosome*

Kelas ini merepresentasikan kromosom dalam algoritma genetika (Subbab ??). Atribut yang terdapat dalam kelas ini adalah:

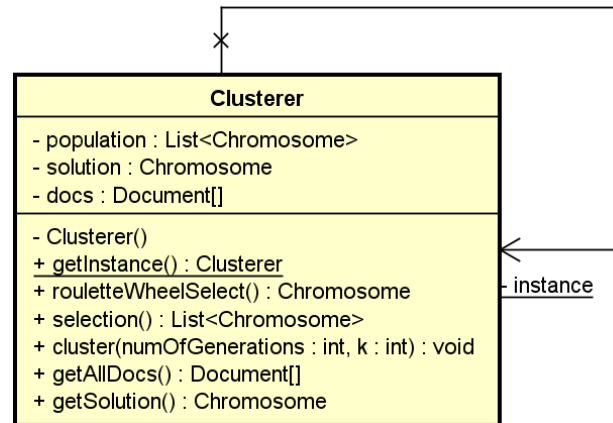
- *genes*: bertipe *List of Gene* dan merupakan kumpulan gen yang terdapat dalam kromosom.
- *MUTATION_PROBABILITY*: merupakan atribut yang bersifat *static* dan *final* yang berisi probabilitas terjadinya mutasi dalam proses pembangkitan keturunan.

Method yang terdapat dalam kelas ini adalah:

- *Chromosome*: merupakan *constructor* tanpa parameter untuk membentuk objek dari kelas *Chromosome*.
- *addGene*: bertugas untuk menambahkan satu gen ke dalam kromosom (ke dalam atribut *genes*).
- *mutate*: berfungsi untuk melakukan mutasi pada kromosom dengan cara melakukan mutasi pada sebuah gen secara acak (Subbab 2.3.4).
- *crossover*: bertugas untuk melakukan persilangan dengan kromosom lain untuk menghasilkan keturunan (Subbab 2.3.3).

- *computeFitness*: mengembalikan nilai *fitness* dari kromosom (Subbab ??).
- *getAllGenes*: merupakan *getter* dari atribut *genes*.

4.1.10 *Clusterer*



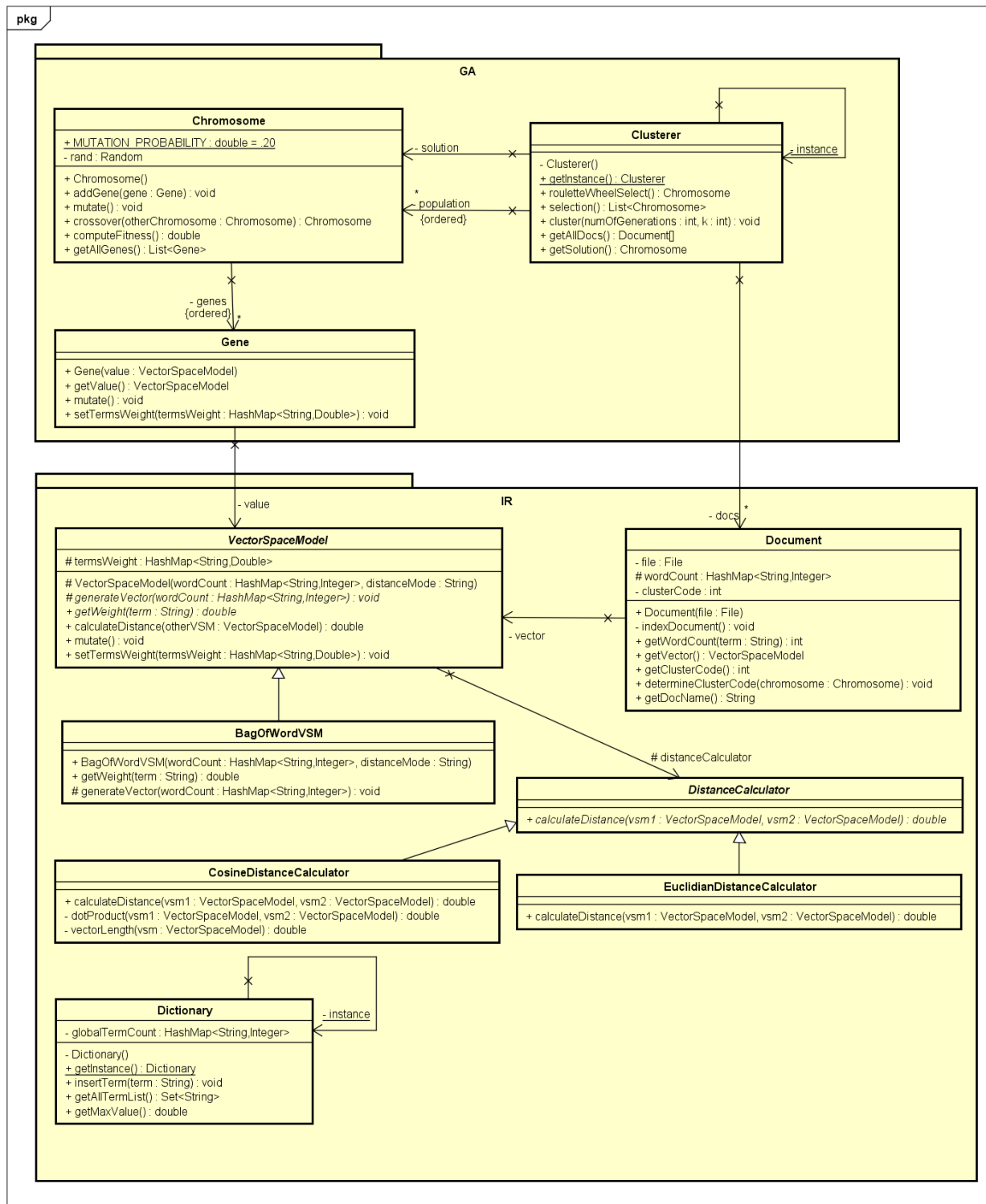
Gambar 4.10: Kelas *Clusterer*

Kelas ini merupakan kelas utama yang akan mengatur jalannya proses pengelompokan. Kelas ini merupakan kelas singleton. Atribut yang terdapat dalam kelas ini adalah:

- *docs*: bertipe *List of Document* yang berfungsi untuk menyimpan seluruh koleksi dokumen.
- *instance*: variabel *static* ini berfungsi untuk menyimpan *instance* dari kelas *Clusterer*.
- *population*: bertipe *List of Chromosome* yang merepresentasikan populasi pada generasi saat ini.
- *solution*: bertipe *Chromosome* yang mencatat kromosom dengan nilai *fitness* terbaik.

Method yang terdapat dalam kelas ini adalah:

- *Clusterer*: merupakan *constructor private* yang berfungsi untuk menjamin tidak akan ada *instance* dibuat diluar dari kelas ini.
- *getInstance*: merupakan *getter* dari variabel *instance*.
- *rouletteWheelSelect*: bertugas untuk memilih dua kromosom dan melakukan persilangan untuk menghasilkan sebuah keturunan selanjutnya.
- *selection*: bertugas untuk melakukan *roulette wheel selection* sebanyak populasi untuk menghasilkan populasi dari generasi selanjutnya.
- *cluster*: merupakan *method* utama yang bertugas melakukan pengelompokan dokumen dengan dua parameter yaitu jumlah generasi dan nilai *k*.
- *getAllDocs*: merupakan *getter* dari atribut *docs*.
- *getSolution*: merupakan *getter* dari atribut *solution*.



Gambar 4.11: Diagram kelas pada tahap perancangan

4.2 Perancangan Antarmuka Pengguna

Antarmuka yang dirancang untuk perangkat lunak ini hanya terdiri dari satu jendela utama dan dua jendela *pop-up*. Pada penelitian ini, perancangan antarmuka dibuat menggunakan perangkat lunak *balsamiq*¹. Setiap objek dan *field* akan diberi label unik agar dapat disesuaikan dengan tabel keterangan. Berikut akan dibahas rancangan antarmuka pengguna dari perangkat ini.

4.2.1 Jendela Utama

Gambar 4.12 merupakan jendela yang pertama kali ditampilkan saat perangkat lunak dijalankan. Jendela tersebut berisi berbagai macam hal yang dibutuhkan pengguna dalam melakukan pengelompokan dokumen.

Gambar 4.12: Rancangan antarmuka jendela utama

Penjelasan setiap *field* dalam jendela utama adalah sebagai berikut:

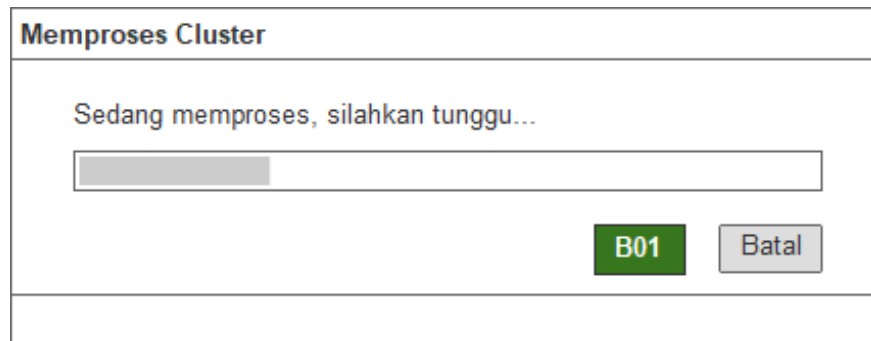
Kode	Nama	Jenis	Default value	Wajib	Aturan validasi
A01	Direktori sumber	<i>file chooser</i>	-	ya	Harus berupa direktori yang berisi dokumen (tidak boleh kosong)
A02	Direktori tujuan	<i>file chooser</i>	-	ya	Harus berupa direktori yang kosong
A03	Jumlah cluster	<i>spinner</i>	2	ya	Nilai minimum 2
A04	Jumlah generasi	<i>spinner</i>	1	ya	Nilai minimum 1
A05	Metode perhitungan jarak	<i>dropdown</i>	<i>Cosine Similarity</i>	ya	-

Tabel 4.1: Rincian *field* pada jendela utama

¹<https://balsamiq.com/>

Jendela ini hanya memiliki sebuah tombol dengan kode **A06** yang berfungsi untuk memulai proses pengelompokan dan membuka jendela *loading*. Selain tombol, pada jendela ini juga terdapat lima *field* seperti yang tertera pada Tabel ??.

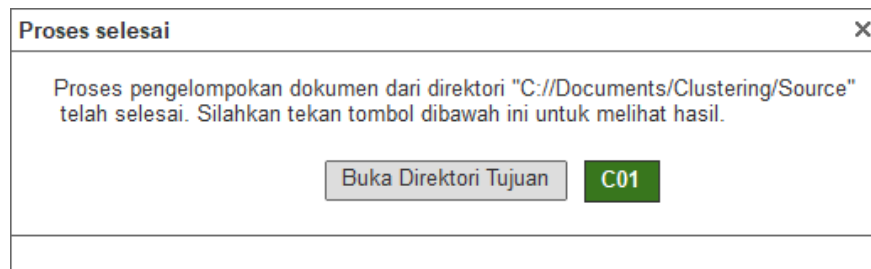
4.2.2 Jendela *Loading*



Gambar 4.13: Rancangan antarmuka jendela *loading*

Gambar 4.13 merupakan jendela yang akan muncul setelah pengguna menekan tombol "Mulai Pengelompokan". Jendela ini berisi informasi perkembangan proses pengelompokan. Informasi ini disajikan dalam bentuk *progress bar*. Hanya ada sebuah tombol pada jendela ini yaitu tombol dengan kode **B01**. Sesuai dengan labelnya, tombol ini berfungsi untuk membatalkan proses pengelompokan. Apabila tombol batal ditekan, maka pengguna akan dikembalikan ke jendela utama (Gambar 4.12).

4.2.3 Jendela Proses Berhasil



Gambar 4.14: Rancangan antarmuka jendela proses berhasil

Gambar 4.14 akan ditampilkan setelah proses pengelompokan berhasil, yaitu saat *progress bar* di jendela *loading* sudah terisi penuh. Jendela ini hanya memiliki satu buah tombol yaitu tombol dengan kode **C01** yang berfungsi untuk membuka *Windows Explorer* pada direktori hasil yang sudah dipilih pengguna pada jendela utama untuk menampilkan hasil dari proses pengelompokan.

DAFTAR REFERENSI

- [1] Raposo, C., Antunes, C. H., dan Barreto, J. P. (2014) Automatic clustering using a genetic algorithm with new solution encoding and operators. *International Conference on Computational Science and Its Applications*, pp. 92–103. Springer.
- [2] Maulik, U. dan Bandyopadhyay, S. (2000) Genetic algorithm-based clustering technique. *Pattern recognition*, **33**, 1455–1465.
- [3] Holland, J. H. (1992) Genetic algorithms. *Scientific american*, **267**, 66–73.
- [4] Sivanandam, S. dan Deepa, S. (2007) *Introduction to Genetic Algorithms*. Springer Science & Business Media.
- [5] Gan, G., Ma, C., dan Wu, J. (2007) *Data clustering: theory, algorithms, and applications*. Siam.
- [6] Zhai, C. dan Massung, S. (2016) *Text data management and analysis: a practical introduction to information retrieval and text mining*. Morgan & Claypool.
- [7] Mecca, G., Raunich, S., dan Pappalardo, A. (2007) A new algorithm for clustering search results. *Data & Knowledge Engineering*, **62**, 504–522.
- [8] Russell, S. J. dan Norvig, P. (2016) *Artificial intelligence: a modern approach*. Malaysia; Pearson Education Limited,.
- [9] Schütze, H., Manning, C. D., dan Raghavan, P. (2008) *Introduction to information retrieval*. Cambridge University Press.
- [10] Aizawa, A. (2003) An information-theoretic perspective of tf-idf measures. *Information Processing & Management*, **39**, 45–65.
- [11] Ahn, C. W. dan Ramakrishna, R. S. (2003) Elitism-based compact genetic algorithms. *IEEE Transactions on Evolutionary Computation*, **7**, 367–385.

LAMPIRAN A

KODE PROGRAM

Listing A.1: MyCode.c

```

1
2 // This does not make algorithmic sense,
3 // but it shows off significant programming characters.
4
5 #include<stdio.h>
6
7 void myFunction( int input, float* output ) {
8     switch ( array[i] ) {
9         case 1: // This is silly code
10             if ( a >= 0 || b <= 3 && c != x )
11                 *output += 0.005 + 20050;
12             char = 'g';
13             b = 2^n + ~right_size - leftSize * MAX_SIZE;
14             c = (--aaa + &daa) / (bbb++ - ccc % 2 );
15             strcpy(a,"hello_$@?");
16         }
17         count = ~mask | 0x00FF00AA;
18     }
19
20 // Fonts for Displaying Program Code in LATEX
21 // Adrian P. Robson, nepsweb.co.uk
22 // 8 October 2012
23 // http://nepsweb.co.uk/docs/progfonts.pdf

```

Listing A.2: MyCode.java

```

1 import java.util.ArrayList;
2 import java.util.Collections;
3 import java.util.HashSet;
4
5 //class for set of vertices close to furthest edge
6 public class MyFurSet {
7     protected int id; //id of the set
8     protected MyEdge FurthestEdge; //the furthest edge
9     protected HashSet<MyVertex> set; //set of vertices close to furthest edge
10    protected ArrayList<ArrayList<Integer>> ordered; //list of all vertices in the set for each trajectory
11    protected ArrayList<Integer> closeID; //store the ID of all vertices
12    protected ArrayList<Double> closeDist; //store the distance of all vertices
13    protected int totaltrj; //total trajectories in the set
14
15    /*
16     * Constructor
17     * @param id : id of the set
18     * @param totaltrj : total number of trajectories in the set
19     * @param FurthestEdge : the furthest edge
20     */
21    public MyFurSet(int id,int totaltrj,MyEdge FurthestEdge) {
22        this.id = id;
23        this.totaltrj = totaltrj;
24        this.FurthestEdge = FurthestEdge;
25        set = new HashSet<MyVertex>();
26        ordered = new ArrayList<ArrayList<Integer>>();
27        for (int i=0;i<totaltrj;i++) ordered.add(new ArrayList<Integer>());
28        closeID = new ArrayList<Integer>(totaltrj);
29        closeDist = new ArrayList<Double>(totaltrj);
30        for (int i = 0;i <totaltrj;i++) {
31            closeID.add(-1);
32            closeDist.add(Double.MAX_VALUE);
33        }
34    }
35
36 }

```


LAMPIRAN B

HASIL EKSPERIMEN

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



Gambar B.1: Hasil 1



Gambar B.2: Hasil 2



Gambar B.3: Hasil 3



Gambar B.4: Hasil 4