SKRIPSI

PENGELOMPOKAN DOKUMEN BERBASIS ALGORITMA GENETIKA

Cornelius David Herianto

NPM: 2015730034

PROGRAM STUDI TEKNIK INFORMATIKA
FAKULTAS TEKNOLOGI INFORMASI DAN SAINS
UNIVERSITAS KATOLIK PARAHYANGAN

«tahun»

UNDERGRADUATE THESIS

GA-BASED DOCUMENT CLUSTERING

Cornelius David Herianto

NPM: 2015730034

DEPARTMENT OF INFORMATICS
FACULTY OF INFORMATION TECHNOLOGY AND SCIENCES
PARAHYANGAN CATHOLIC UNIVERSITY
«tahun»

LEMBAR PENGESAHAN

PENGELOMPOKAN DOKUMEN BERBASIS ALGORITMA GENETIKA

Cornelius David Herianto

NPM: 2015730034

Bandung, «tanggal» «bulan» «tahun»

Menyetujui,

Pembimbing

Dott. Thomas Anung Basuki

Ketua Tim Penguji	Anggota Tim Penguji
«penguji 1»	«penguji 2»

Mengetahui,

Ketua Program Studi

Mariskha Tri Adithia, P.D.Eng

PERNYATAAN

Dengan ini saya yang bertandatangan di bawah ini menyatakan bahwa skripsi dengan judul:

PENGELOMPOKAN DOKUMEN BERBASIS ALGORITMA GENETIKA

adalah benar-benar karya saya sendiri, dan saya tidak melakukan penjiplakan atau pengutipan dengan cara-cara yang tidak sesuai dengan etika keilmuan yang berlaku dalam masyarakat keilmuan.

Atas pernyataan ini, saya siap menanggung segala risiko dan sanksi yang dijatuhkan kepada saya, apabila di kemudian hari ditemukan adanya pelanggaran terhadap etika keilmuan dalam karya saya, atau jika ada tuntutan formal atau non-formal dari pihak lain berkaitan dengan keaslian karya saya ini.

Dinyatakan di Bandung,
Tanggal «tanggal» «bulan» «tahun»

Meterai
Rp. 6000

Cornelius David Herianto
NPM: 2015730034

ABSTRAK

«Tuliskan abstrak anda di sini, dalam bahasa Indonesia»

Kata-kata kunci: «Tuliskan di sini kata-kata kunci yang anda gunakan, dalam bahasa Indonesia»

ABSTRACT

«Tuliskan abstrak anda di sini, dalam bahasa Inggris»

Keywords: «Tuliskan di sini kata-kata kunci yang anda gunakan, dalam bahasa Inggris»

«kepada siapa anda mempersembahkan skripsi ini. . . ?»

KATA PENGANTAR

«Tuliskan kata pengantar dari anda di sini . . . »

Bandung, «bulan» «tahun»

Penulis

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

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 masukkan 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. Pe-ngelompokan dalam informatika merupakan bagian dari bidang pembelajaran mesin. Dalam pembelajaran mesin, terdapat dua jenis pengelompokan, 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*.

Domain masalah dalam penelitian ini adalah dokumen sehingga sangat erat kaitannya dengan bidang temu kembali informasi (*information retrieval*). Sebelum dapat diolah lebih lanjut, dokumen-dokumen yang telah ada harus diolah sehingga menjadi suatu bentuk dengan nilai informasi yang dapat dimanipulasi oleh komputer. Biasanya dokumen akan direpresentasikan ke dalam bentuk vektor yang disebut dengan model ruang vektor.

2.1 Temu Kembali Informasi

Temu kembali informasi adalah bidang ilmu yang berurusan dengan representasi, penyimpanan, pengolahan, dan akses terhadap informasi.[5] Pada penelitian ini, temu kembali informasi memiliki peran untuk mengubah dokumen yang semula berbentuk teks menjadi sesuatu yang memiliki nilai informasi agar nantinya bisa diproses lebih lanjut (dalam kasus ini dengan pengelompokan dokumen).

Dalam penelitian ini, temu kembali informasi digunakan untuk merepresentasikan dokumen ke dalam model ruang vektor agar bisa dilakukan pengelompokan. Tahap pertama yang dilakukan adalah membentuk kosa kata (*vocabulary*) yang berisi seluruh istilah berbeda yang ada di setiap dokumen yang akan diindeks. Kemudian, untuk setiap dokumen akan dibentuk suatu indeks yang terdiri dari pasangan antara istilah di dokumen tersebut dan jumlah kemunculannya (*term-document incidence matrix*). Tabel 2.1.1 merupakan contoh dari *term-document incidence matrix* dengan baris pada tabel menunjukkan istilah (*term*) dan kolom menunjukkan nama dokumen. Sel bernilai 1 apabila dokumen mengandung term tertentu dan 0 jika tidak.

Selanjutnya, setiap jumlah kemunculan suatu istilah dalam sebuah dokumen akan diubah menjadi suatu bobot tertentu berdasarkan teknik pemodelannya. Dalam penelitian ini, digunakan 2 teknik pembobotan yaitu bobot frekuensi dan bobot tf-idf.

2.1.1 Bobot frekuensi

Bobot frekuensi merupakan teknik pembobotan yang sangat sederhana karena bobotnya merupakan jumlah kemunculan istilah tersebut dalam dokumen. Bobot frekuensi dapat digambarkan dengan Persamaan 2.1

$$w_i = tf_i \tag{2.1}$$

dengan w_i merupakan bobot istilah ke-i dan tf_i merupakan frekuensi kemunculan istilah ke-i pada dokumen.

	Anthony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth	...
Anthony	1	1	0	0	0	1	
Brutus	1	1	0	1	0	0	
Caesar	1	1	0	1	1	1	
Calpurnia	0	1	0	0	0	0	
Cleopatra	1	0	0	0	0	0	
mercy	1	0	1	1	1	1	
worser	1	0	1	1	1	0	
...							

Tabel 2.1: *Term-document incidence matrix*

2.1.2 Bobot TF-IDF

Pada bobot frekuensi, bobot hanya dihitung berdasarkan kemunculan istilah dalam dokumen itu sendiri. Namun dalam bobot TF-IDF (*term frequency-inverse document frequency*), bobot juga dihitung berdasarkan kemunculan istilah pada himpunan dokumen. Metode ini sangat populer digunakan oleh sistem rekomendasi berbasis teks.[6] Rumus dari pembobotan menggunakan TF-IDF ditunjukkan oleh Persamaan 2.2

$$w_i = tf_i \times log\frac{N}{N_i} \tag{2.2}$$

dengan w_i merupakan bobot istilah ke-i, tf_i merupakan frekuensi kemunculan istilah ke-i pada dokumen, N menyatakan banyaknya anggota himpunan dokumen, dan N_i menunjukkan frekuensi dokumen dari istilah ke-i (banyaknya dokumen pada himpunan dokumen yang memuat istilah ke-i).

Berdasarkan rumus tersebut, maka dapat ditarik dua kesimpulan yaitu:

- Semakin sering suatu istilah muncul di suatu dokumen, maka semakin representatif istilah tersebut terhadap isi dokumen.
- Semakin banyak dokumen yang memuat suatu istilah, maka nilai informasi istilah tersebut semakin kecil.

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

2.1.3 Model ruang vektor

Model ruang vektor adalah representasi dari koleksi dokumen sebagai vektor dalam ruang vektor yang umum.[7] Model ruang vektor ini biasanya digunakan dalam sejumlah operasi pencarian informasi mulai dari penilaian dokumen pada *query*, klasifikasi dokumen dan pengelompokan dokumen.

Dalam pengolahan model ruang vektor, dibutuhkan cara untuk menghitung kesamaan antara dua ruang vektor. Dalam pengelompokan dokumen, apabila kesamaan antara dua buah vektor semakin besar, maka peluang kedua vektor tersebut berada dalam sebuah kelompok yang sama akan semakin besar. Dalam penelitian ini, digunakan dua cara untuk menghitung kesamaan antara dua ruang vektor yaitu menggunakan Jarak Euclidean (*Euclidean distance*) dan persamaan cosinus (*cosine similarity*).

Jarak Euclidean

Jarak Euclidean atau biasa disebut jarak garis lurus merupakan metode paling banyak digunakan untuk menghitung jarak antara dua buah vektor. Secara umum, rumus dari Jarak Euclidean ditunjukkan oleh persamaan 2.3

$$d_{ij} = \sqrt{\sum_{v=1}^{N}(x_{vi} - x_{vj})^2} \tag{2.3}$$

dengan d_{ij} adalah jarak antara vektor ke-i dengan vektor ke-j, N adalah jumlah dimensi pada vektor, dan x_{vi} adalah nilai dimensi ke-v dari vektor ke-i.

Persamaan Cosinus

Persamaan cosinus merupakan normalisasi hasil kali titik dengan panjang masing-masing vektor. Persamaan cosinus ditunjukkan dalam persamaan 2.4

$$s_{ij} = \frac{i \cdot j}{\| i \| \times \| j \|} \tag{2.4}$$

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.

2.2 Pembelajaran Mesin

Pembelajaran mesin merupakan sistem yang dapat beradaptasi dengan keadaan baru dan dapat mengenali pola [8]. Sistem yang telah belajar akan meningkatkan kinerjanya pada tugas-tugas di masa yang akan datang setelah melakukan pengamatan tentang lingkungannya.

2.2.1 Pengelompokan

Tugas pengelompokan (*clustering*) merupakan salah satu bagian dari pembelajaran mesin. Pengelompokan terdiri dari dua jenis berdasarkan metode pembelajarannya. *Classification* merupakan jenis pembelajaran terarah karena sudah diberikan label sejak awal, lalu dilakukan pengelompokan berdasarkan label untuk setiap kelompoknya. Sedangkan *clustering* merupakan jenis pembelajaran tak terarah karena tidak diberikan label sejak awal sehingga pengelompokan dilakukan berdasarkan suatu kesamaan tertentu.

K-Means

K-means merupakan algoritma pengelompokan yang paling populer digunakan saat ini. algoritma ini membagi data ke dalam K *cluster*. Setiap *cluster* direpresentasikan dengan titik tengahnya (*centroid*). Setiap iterasi, titik tengah akan dihitung sebagai rata-rata dari semua titik data dari *cluster* tersebut. Persamaaan 2.5 merupakan persamaan untuk menghitung *centroid*

$$\mu_i = \frac{1}{N_i}\sum_{q=1}^{N_i} x_q \tag{2.5}$$

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

Algoritma K-means diawali dengan penentuan *centroid* secara acak. Pada setiap iterasi, setiap data ditetapkan pada *cluster* yang memiliki *centroid* dengan jarak terdekat dari titik data tersebut, kemudian posisi *centroid* dari setiap *cluster* akan dihitung ulang dengan Persamaan 2.5. Iterasi akan terus diulang sampai posisi dari semua *cluster* tidak berubah.

Algorithm 1 K-Means

Input: S (himpunan titik data), K (Jumlah *cluster*)
Output: himpunan *cluster*

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

2.3 Algoritma Genetika

Algoritma genetika (GA) adalah suatu algoritma pencarian yang terinspirasi dari proses seleksi alam yang terjadi secara alami dalam proses evolusi. Ada beberapa istilah yang digunakan dalam algoritma genetika di antaranya kromosom, seleksi, persilangan, mutasi, dan fungsi *fitness*.

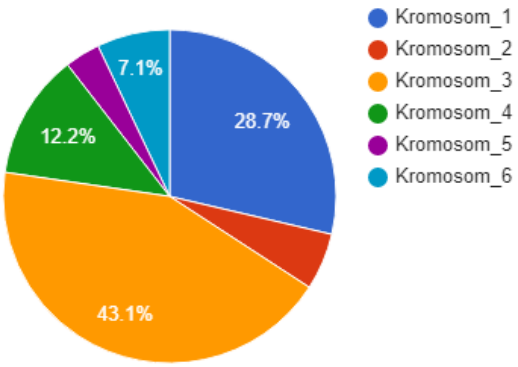
2.3.1 Kromosom

Dalam GA, kromosom adalah himpunan parameter yang mendefinisikan suatu solusi yang diusulkan. Kromosom biasanya direpresentasikan sebagai string yang berisi kumpulan nilai (gen), meskipun berbagai struktur data lainnya juga digunakan.

2.3.2 Seleksi

Seleksi dalam algoritma genetika bertugas untuk memilih kromosom dari populasi untuk proses persilangan berdasarkan nilai *fitness*. 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 2.1 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.

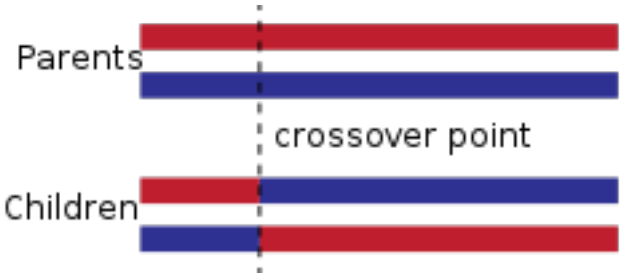
Dalam GA, proses seleksi akan memilih sejumlah induk yang cukup untuk reproduksi dan membentuk generasi selanjutnya. Dalam penelitian ini diterapkan strategi *elitism* yaitu dengan langsung memindahkan individu dengan nilai *fitness* terbesar ke generasi selanjutnya. Hal ini dilakukan untuk mencegah individu tersebut terhilang dari populasi dalam proses reproduksi.



Gambar 2.1: Diagram Lingkaran

2.3.3 Persilangan

Persilangan adalah operasi genetik yang digunakan untuk menggabungkan informasi genetik dari dua induk untuk menghasilkan keturunan baru. Teknik persilangan yang digunakan dalam penelitian ini adalah *Single-point crossover*. Dalam teknik ini, 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 2.2.



Gambar 2.2: *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. Mutasi mengubah satu atau beberapa nilai dalam gen. Mutasi terjadi berdasarkan probabilitas mutasi yang sudah ditentukan sebelumnya. Probabilitas ini seharusnya bernilai kecil, karena jika terlalu besar maka akan menjadi sama dengan algoritma pencarian acak primitif (*primitive random search*). Pada penelitian ini, kromosom memiliki gen yang bernilai non-biner. Oleh karena itu, mutasi dilakukan dengan memilih sebuah bilangan acak antara batas bawah dan batas atas nilai suatu gen.

2.3.5 Fungsi *Fitness*

Fungsi *fitness* adalah fungsi objektif yang digunakan untuk mengukur seberapa baik suatu kromosom sebagai calon solusi. Sebuah fungsi *fitness* harus bisa memperkirakan seberapa dekat sebuah calon solusi dengan solusi yang optimal. Dalam algoritma genetika, fungsi *fitness* juga digunakan dalam proses seleksi (*roulette-wheel selection*) untuk menentukan seberapa baik suatu kromosom untuk menjadi induk dari generasi berikutnya.

2.3.6 Proses pencarian dalam algoritma genetika

Seperti yang disebutkan dalam Algoritma 2, algoritma genetika dimulai dengan menginisialisasi suatu populasi (biasanya dibangkitkan secara acak jika tidak diketahui heuristik masalahnya). Lalu, akan dibangkitkan populasi baru untuk generasi berikutnya dengan cara mengambil dua kromosom secara acak dengan teknik *roulette-wheel selection*. Setelah itu akan dilakukan persilangan dengan teknik *single-point crossover*. Teknik ini dilakukan dengan cara menentukan sebuah titik potong c yang diambil secara acak antara angka 1 sampai panjang kromosom n. Keturunan dari kedua induk tersebut akan memiliki kromosom induk pertama dari gen ke-1 sampai gen ke-c dan dari induk kedua mulai dari gen ke-(c+1) sampai gen ke-n. Setelah itu, apabila terjadi mutasi, maka salah satu gen dari anak akan diubah nilainya. Iterasi ini akan dilakukan terus-menerus hingga kriteria berhenti tercapai atau sudah mencapai batas jumlah generasi tertentu.

2.4 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 karena 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

solusi ← himpunan solusi tiap generasi
repeat
  tambahkan individu dengan nilai *fitness* tertinggi dari *populasi* ke *solusi*
  populasi_baru ← himpunan kosong
  for i=1 **to** Size(*populasi*) **do**
    x ← Seleksi-acak(*populasi*)
    y ← Seleksi-acak(*populasi*)
    anak ← Persilangan(x, y)
    rand ← Random(0,1)
    if *rand* ≤ *prob_mutasi* **then**
      anak ← Mutasi(*anak*)
    end if
    tambahkan *anak* ke *populasi_baru*
  end for
  populasi ← *populasi_baru*
until N solusi terakhir pada *solusi* tidak memiliki perubahan yang signifikan
return individu terbaik dalam populasi, berdasarkan nilai *fitness*

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

inputs: x dan y, individu induk

n ← Length(x)
c ← angka acak antara 1 sampai n
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

```
sum ← 0
for all individu ∈ populasi do
    sum ← sum + Fitness(individu)
end for
terpilih ← Random(0,1) × sum
for all individu ∈ populasi do
    terpilih ← terpilih − Fitness(individu)
    if terpilih ≤ 0 then
        return individu
    end if
end for
return individu dengan urutan terakhir di populasi
```

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

inputs: *individu*, individu yang akan dilakukan mutasi

```
n ← Length(x)
c ← angka acak antara 1 sampai n
ubah nilai gen ke-c pada individu {nilai bervariasi tergantung metode}
return individu
```

BAB 3

ANALISIS

Untuk menyelesaikan masalah pengelompokan dokumen menggunakan algoritma genetika, maka perlu dibangun sebuah model yang dapat diterapkan ke dalam algoritma tersebut.

3.1 Representasi Kromosom

Setiap *string* kromosom merupakan deretan bilangan riil yang merepresentasikan K titik pusat *cluster* (*centroid*). 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, dalam suatu bidang dua dimensi akan dilakukan pengelompokan ke dalam tiga *cluster* $C1, C2$, dan $C3$. Pada suatu iterasi, $C1$ berada pada posisi (3.05, 1.43), $C2$ berada pada posisi (15.85, 14.23), dan $C3$ berada pada posisi (5.12, 9.45). Kromosom yang terbentuk dari ketiga *cluster* ditunjukkan dalam Tabel 3.1.

C1		**C2**		**C3**	
3.05	1.43	15.85	14.23	5.12	9.45

Tabel 3.1: Representasi *centroid* ke dalam kromosom

3.2 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 $x_i, i = 1, 2, ..., n$ ke dalam sebuah *cluster* C_j dengan *centroid* z_j sehingga

$$\|x_i - z_j\| < \|x_i - z_p\|, p = 1, 2, ..., K, \text{ dan } p \neq j. \tag{3.1}$$

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 dihitung menggunakan persamaan 3.2

$$z_i^* = \frac{1}{n_i} \sum_{x_j \in C_i} x_j, i = 1, 2, ..., K. \tag{3.2}$$

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

3.2.1 Euclidean Distance

Pada perhitungan dengan *euclidean distance*, akan dihitung sebuah *clustering metric* M dengan persamaan 3.3

$$\begin{aligned} M &= \sum_{i=1}^{K} M_i, \\ M_i &= \sum_{x_j \in C_i} \| x_j - z_i \| \end{aligned} \tag{3.3}$$

Lalu, fungsi *fitness* akan didefinisikan sebagai $f = 1/M$, sehingga maksimalisasi terhadap nilai f akan meminimalkan nilai M.

3.2.2 Cosine Similarity

Perhitungan *fitness* menggunakan *cosine similarity* dapat dilakukan dengan persamaan 3.4

$$\begin{aligned} f &= \sum_{i=1}^{K} f_i, \\ f_i &= \sum_{x_j \in C_i} \frac{x_j \cdot z_i}{\| x_j \| \times \| z_i \|} \end{aligned} \tag{3.4}$$

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

3.3 Operasi Genetik

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

3.3.1 Inisialisasi Populasi

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

3.3.2 Seleksi

Proses seleksi ini terjadi berdasarkan konsep *survival of the fittest* yang diadaptasi dari sistem genetika alami. Konsep ini mengatakan bahwa hanya individu terbaik yang akan bertahan hidup (lolos seleksi alam). Dengan mengadaptasi konsep tersebut, GA menerapkan seleksi berdasarkan nilai *fitness* tiap individu. Dalam penelitian ini, calon induk dari generasi selanjutnya dipilih dengan menggunakan teknik *roulette-wheel selection*. Seperti yang telah dijelaskan dalam subbab 2.3.2, dalam teknik *roulette-wheel selection* individu yang memiliki nilai *fitness* lebih tinggi akan memiliki peluang lebih besar untuk terpilih dan menjadi induk bagi generasi berikutnya. Namun karena masih ada peluang individu dengan nilai *fitness* terbaik tidak terpilih, maka dalam penelitian ini juga akan digunakan teknik elitisme [9]. Dengan digunakannya elitisme, maka individu dengan nilai *fitness* terbaik akan disalin dan langsung menjadi anggota dari generasi berikutnya. Hal ini akan menjamin individu terbaik tidak akan hilang akibat tidak terpilih oleh *roulette-wheel selection*.

3.3.3 Persilangan

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

3.3.4 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 3.1, 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 3.2

C1		**C2**		**C3**	
3.05	**4.18**	15.85	14.23	5.12	9.45

Tabel 3.2: Kromosom hasil mutasi

BAB 4

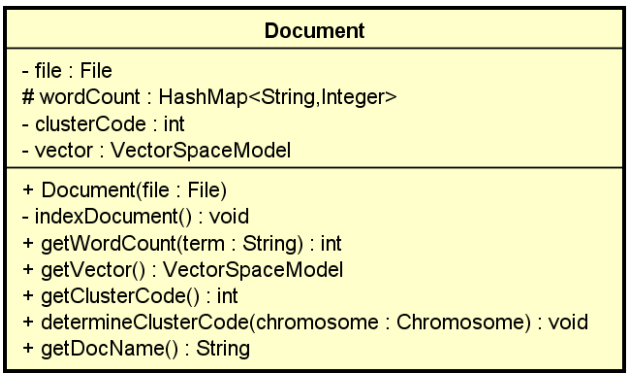
PERANCANGAN

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:

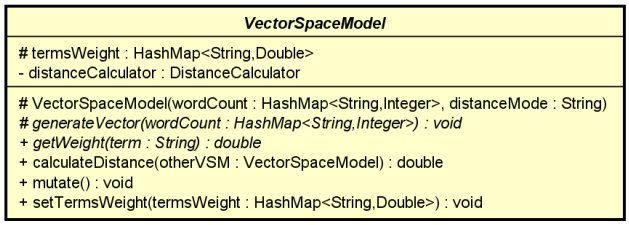
- *wordCount*: bertipe *HashMap* dengan *key* bertipe *String* dan *value* bertipe *Integer*. Atribut ini menyimpan pasangan kata yang dimiliki oleh dokumen tersebut dan frekuensinya.
- *file*: atribut ini bertipe *File* milik *package* java.io yang berfungsi untuk merepresentasikan *file* dari dokumen 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*



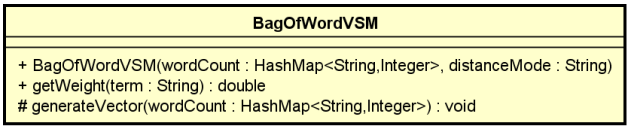
Gambar 4.2: Kelas *VectorSpaceModel*

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 bebatnya 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*.

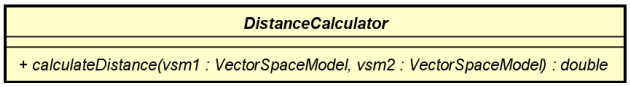


Gambar 4.3: Kelas *BagOfWordVSM*

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.

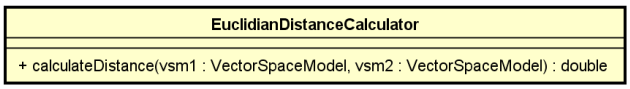
4.1.4 *DistanceCalculator*



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*



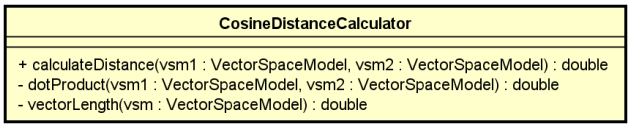
Gambar 4.5: Kelas *EuclideanDistanceCalculator*

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 2.1.3).

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:

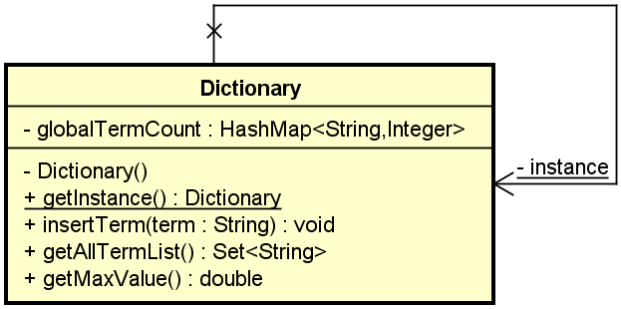
- *calculateDistance*: merupakan *method* yang diturunkan dari kelas *DistanceCalculator*. *Method* ini mengembalikan jarak dari *vsm1* dan *vsm2* yang dihitung menggunakan persamaan cosinus (Subbab 2.1.3).



Gambar 4.6: Kelas *CosineDistanceCalculator*

- *dotProduct*: berfungsi untuk menghitung hasil perkalian titik (*dot product*) antara *vsm1* dan *vsm 2*.
- *vectorLength*: berfungsi untuk menghitung panjang dari vektor *vsm*.

4.1.7 *Dictionary*



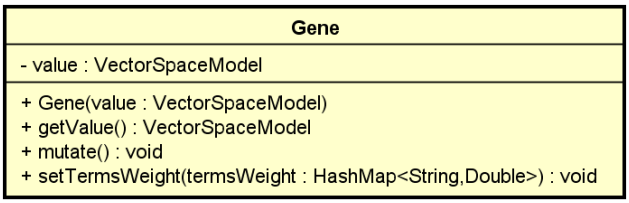
Gambar 4.7: Kelas *Dictionary*

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.



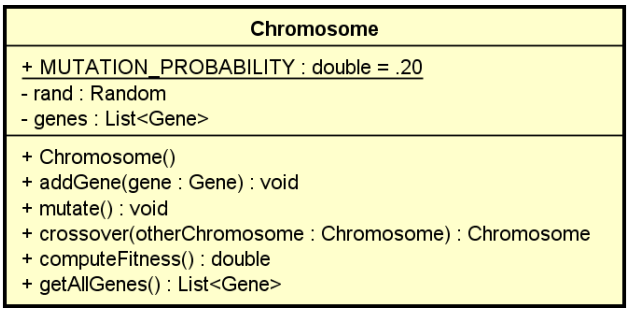
Gambar 4.8: Kelas *Gene*

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*: 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*



Gambar 4.9: Kelas *Chromosome*

Kelas ini merepresentasikan kromosom dalam algoritma genetika (Subbab 2.3.1). Atribut yang terdapat dalam kelas ini adalah:

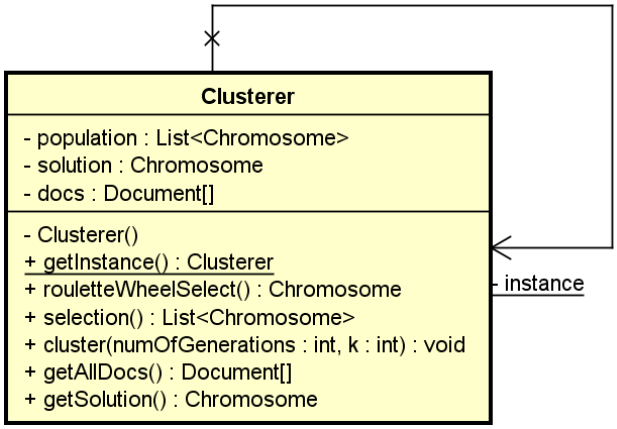
- *genes*: bertipe *List of Gene* dan merupakan kumpulan gen yang terdapat dalam kromosom.
- *MUTATION_ PROBABILITY*: merupakan atribut yang bersifat *static* dan *final* yang berisi probabilitas terjadinya mutasi dalam proses pembangkitan keturunan.

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 mengahsilkan keturunan (Subbab 2.3.3).
- *computeFitness*: mengembalikan nilai *fitness* dari kromosom (Subbab 2.3.5).
- *getAllGenes*: merupakan *getter* dari atribut *genes*.

4.1.10 *Clusterer*



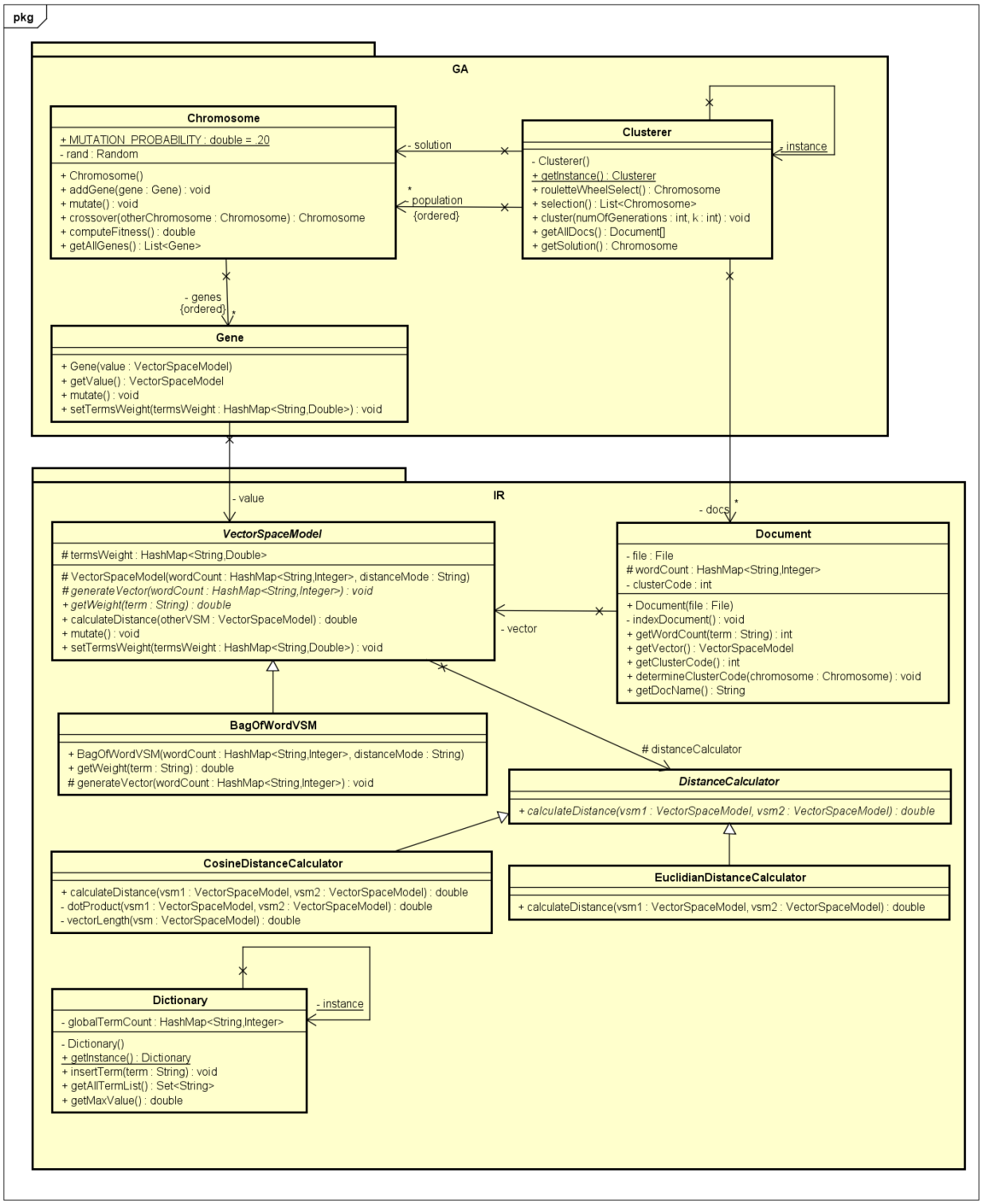
Gambar 4.10: Kelas *Clusterer*

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*.



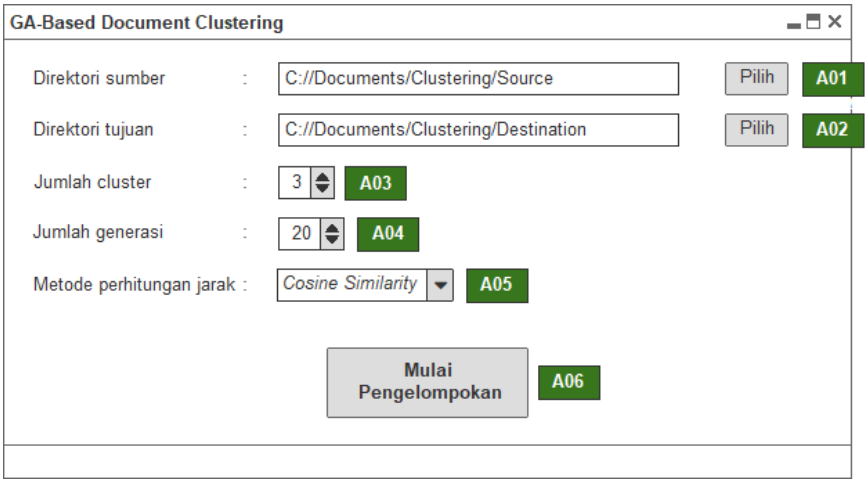
Gambar 4.11: Diagram kelas pada tahap perancangan

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 penguna dalam melakukan pengelompokan dokumen.



Gambar 4.12: Rancangan antarmuka jendela utama

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

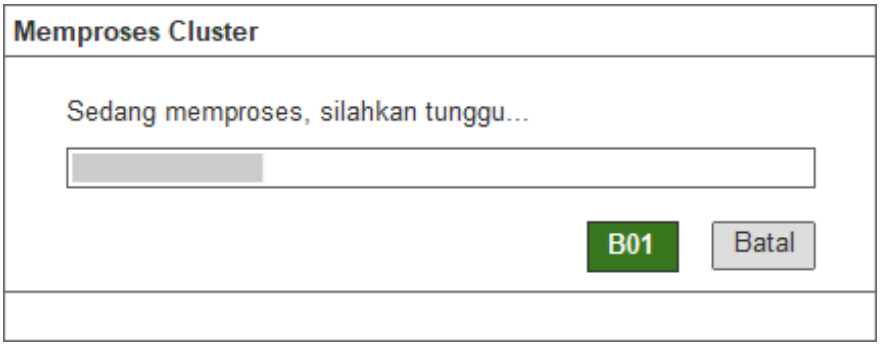
Kode	Nama	Jenis	*Defaut value*	Wajib	Aturan validasi
A01	Direktori sumber	*file chooser*	-	ya	Harus berupa direktori yang berisi dokumen (tidak boleh kosong)
A02	Direktori tujuan	*file chooser*	-	ya	Harus berupa direktori yang kosong
A03	Jumlah cluster	*spinner*	2	ya	Nilai minimum 2
A04	Jumlah generasi	*spinner*	1	ya	Nilai minimum 1
A05	Metode perhitungan jarak	*dropdown*	*Cosine Similarity*	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 4.2.1.

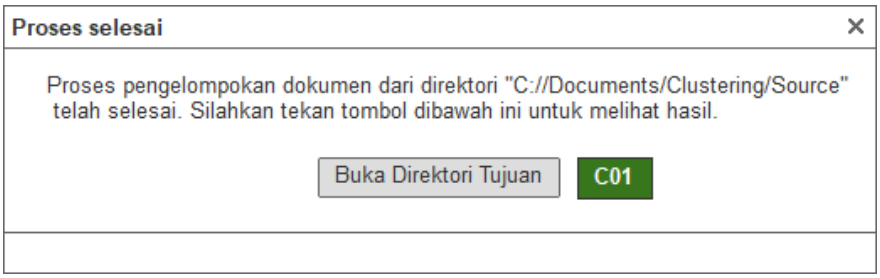
4.2.2 Jendela *Loading*



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] Baeza-Yates, R., Ribeiro-Neto, B., dkk. (1999) *Modern information retrieval*. ACM press New York.

[6] Aizawa, A. (2003) An information-theoretic perspective of tf–idf measures. *Information Processing & Management*, **39**, 45–65.

[7] Schütze, H., Manning, C. D., dan Raghavan, P. (2008) *Introduction to information retrieval*. Cambridge University Press.

[8] Russell, S. J. dan Norvig, P. (2016) *Artificial intelligence: a modern approach*. Malaysia; Pearson Education Limited,.

[9] 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

```c

// This does not make algorithmic sense,
// but it shows off significant programming characters.

#include<stdio.h>

void myFunction( int input, float* output ) {
    switch ( array[i] ) {
        case 1: // This is silly code
            if ( a >= 0 || b <= 3 && c != x )
                *output += 0.005 + 20050;
            char = 'g';
            b = 2^n + ~right_size - leftSize * MAX_SIZE;
            c = (--aaa + &daa) / (bbb++ - ccc % 2 );
            strcpy(a,"hello $@?");
    }
    count = ~mask | 0x00FF00AA;
}

// Fonts for Displaying Program Code in LATEX
// Adrian P. Robson, nepsweb.co.uk
// 8 October 2012
// http://nepsweb.co.uk/docs/progfonts.pdf
```

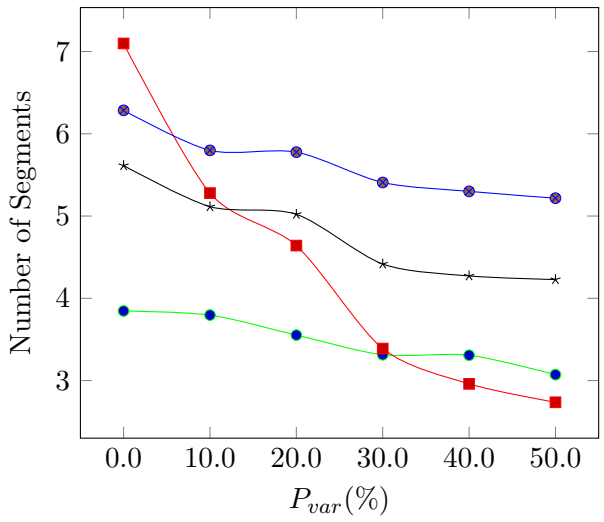
Listing A.2: MyCode.java

```java
import java.util.ArrayList;
import java.util.Collections;
import java.util.HashSet;

//class for set of vertices close to furthest edge
public class MyFurSet {
    protected int id;                                   //id of the set
    protected MyEdge FurthestEdge;                      //the furthest edge
    protected HashSet<MyVertex> set;                    //set of vertices close to furthest edge
    protected ArrayList<ArrayList<Integer>> ordered;    //list of all vertices in the set for each trajectory
    protected ArrayList<Integer> closeID;               //store the ID of all vertices
    protected ArrayList<Double> closeDist;              //store the distance of all vertices
    protected int totaltrj;                             //total trajectories in the set

    /*
     * Constructor
     * @param id : id of the set
     * @param totaltrj : total number of trajectories in the set
     * @param FurthestEdge : the furthest edge
     */
    public MyFurSet(int id,int totaltrj,MyEdge FurthestEdge) {
        this.id = id;
        this.totaltrj = totaltrj;
        this.FurthestEdge = FurthestEdge;
        set = new HashSet<MyVertex>();
        ordered = new ArrayList<ArrayList<Integer>>();
        for (int i=0;i<totaltrj;i++) ordered.add(new ArrayList<Integer>());
        closeID = new ArrayList<Integer>(totaltrj);
        closeDist = new ArrayList<Double>(totaltrj);
        for (int i = 0;i <totaltrj;i++) {
            closeID.add(-1);
            closeDist.add(Double.MAX_VALUE);
        }
    }

}
```

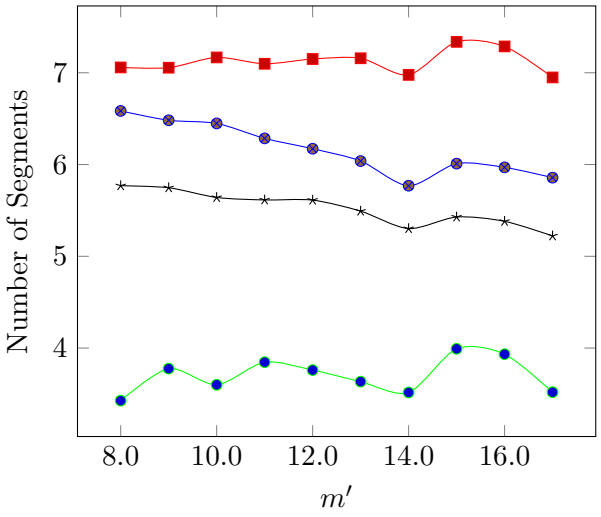
LAMPIRAN B

HASIL EKSPERIMEN

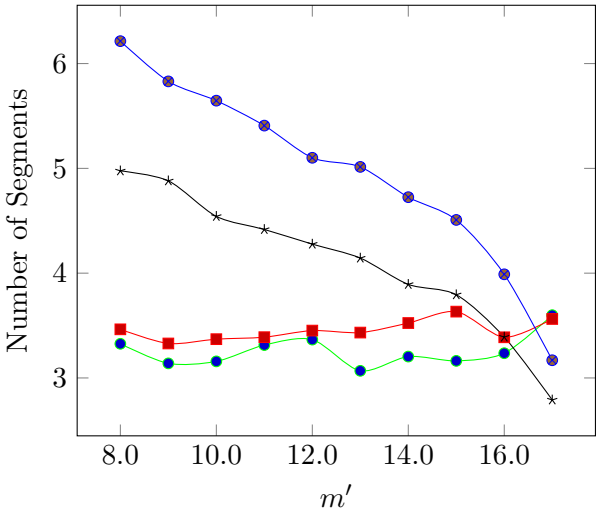
Hasil eksperimen berikut dibuat dengan menggunakan TIKZPICTURE (bukan hasil excel yg diubah ke file bitmap). Sangat berguna jika ingin menampilkan tabel (yang kuantitasnya sangat banyak) yang datanya dihasilkan dari program komputer.



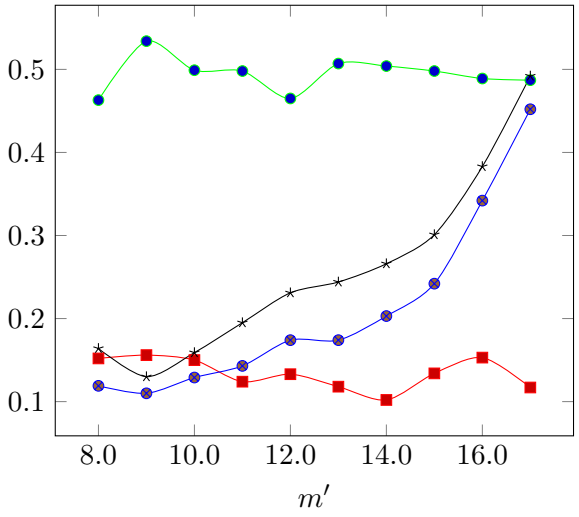
Gambar B.1: Hasil 1



Gambar B.2: Hasil 2



Gambar B.3: Hasil 3



Gambar B.4: Hasil 4