SKRIPSI

PENGELOMPOKAN DOKUMEN BERBASIS ALGORITMA GENETIKA

Cornelius David Herianto

NPM: 2015730034

PROGRAM STUDI TEKNIK INFORMATIKA
FAKULTAS TEKNOLOGI INFORMASI DAN SAINS
UNIVERSITAS KATOLIK PARAHYANGAN
2019

UNDERGRADUATE THESIS

GA-BASED DOCUMENT CLUSTERING

Cornelius David Herianto

NPM: 2015730034

DEPARTMENT OF INFORMATICS
FACULTY OF INFORMATION TECHNOLOGY AND SCIENCES
PARAHYANGAN CATHOLIC UNIVERSITY
2019

ABSTRAK

Pengelompokan (*clustering*) merupakan sebuah metode untuk menggabungkan himpunan objek ke dalam kelompok-kelompok sedemikan rupa sehingga objek dalam kelompok (*cluster*) lebih mirip (karena suatu hal) satu sama lain daripada objek di kelompok lain [5]. Salah satu algoritma pengelompokan yang paling sering digunakan adalah *K-means*. Namun, algoritma *K-means* memiliki kekurangan yaitu dapat terjebak dalam *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*).

Algoritma genetika atau biasa disebut *Genetic Algorithm* (GA) adalah suatu algoritma pencarian yang terinspirasi dari proses seleksi alam yang terjadi secara alami dalam proses evolusi. GA merupakan metode penyelesaian masalah yang menggunakan genetika sebagai pemodelannya. Suatu calon solusi dalam GA dimodelkan sebagai suatu individu. Kumpulan individu-individu ini disebut dengan populasi. Setiap individu dalam populasi direpresentasikan dengan kromosom. Kromosom merupakan kumpulan parameter yang membentuk suatu solusi. Parameter-parameter yang menyusun kromosom disebut dengan gen. Setiap kromosom memiliki suatu nilai yang terkait dengan *fitness* dari solusi yang direpresentasikannya. Nilai itu biasanya disebut dengan nilai *fitness*.

Dalam penelitian ini, GA akan digunakan sebagai solusi dari masalah *local optimum*. *Local optimum* dapat diatasi oleh GA yang sudah terbukti efektif dalam masalah pencarian dan optimasi. Dalam penelitian ini, GA dapat digunakan untuk mengelompokkan dokumen dengan beberapa adaptasi terhadap representasi kromosom, fungsi *fitness*, seleksi, persilangan, dan mutasi. Algoritma genetika dan algoritma *K-means* akan diuji menggunakan suatu *dataset* berlabel untuk membandingkan waktu dan hasil pengelompokan dari kedua algoritma tersebut.

Kata-kata kunci: Algoritma genetika, Pengelompokan dokumen, Algoritma *K-means*, TF-IDF, *Local optimum*

ABSTRACT

Clustering is a method of creating groups of objects, or clusters, in such a way that objects in one cluster are very similar and objects in different clusters are quite distinct [5]. One of the most frequently used algorithm in clustering is K-means. However, K-means can easily stuck in local optimum. Local optimum of an optimization problem is a solution that is optimal (either maximal or minimal) within a neighboring set of candidate solutions. This is in contrast to a global optimum, which is the optimal solution among all possible solutions, not just those in a particular neighborhood of values.

Genetic Algorithm (GA) is a search algorithm inspired by the natural selection process that occurs naturally in the evolutionary process. GA is a problem solving method that used genetics as its model. A solution candidate is modelled as an individual in GA. Set of these individuals are called population. Every individual in a population is represented by a chromosome. Chromosome is a collection of parameters which formed a solution. That parameters is called gene. Every individual in the population is assigned, by means of a fitness function, a measure of its goodness

In this study, GA will be used as a solution to local optimum, which have proven to be effective in search and optimization problems. GA can be used to cluster documents by adapting chromosome representation, fitness function, selection, crossover, and mutation. Genetic algorithm and K-means algorithm will be tested using a labeled dataset to compare their running time and clustering result.

Keywords: Genetic algorithm, Document clustering, K-means algorithm, TF-IDF, Local optimum

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 Landasan Teori
   Bab 2 memuat landasan teori yang digunakan dalam penelitian ini. Konsep-konsep yang dibahas yaitu pengelompokan, *local optimum*, K-means, algoritma genetika beserta seluruh operasinya, model ruang vektor, pembobotan term yang terdiri dari bobot frekuensi dan bobot TF-IDF, metrik *Intracluster* untuk mengukur minerja metode *clustering*, dan rvaluasi hasil pengelompokan menggunakan *purity*.

3. Bab 3 Analisis
   Bab 3 memuat hasil analisis berdasarkan landasan teori. Hasil analisis yang ditulis pada bab 3 antara lain analisis *dataset*, representasi dokumen, modifikasi terhadap model ruang vektor beserta metode pembobotannya, representasi kromosom, fungsi *fitness*, dan operasi genetik lainnya.

4. Bab 4 Perancangan
   Bab 4 memuat hasil perancangan berdasarkan hasil analisis pada bab 3. Terdapat tiga bagian dalam bab perancangan yaitu Kebutuhan masukan dan keluaran, perancangan kelas, dan perancangan antarmuka pengguna.

5. Bab 5 Pengujian dan Eksperimen
   Bab 5 memuat hasil pengujian dan eksperimen yang telah dilakukan. Pada bab ini dibahas mengenai skenario pengujian, eksperimen pada algoritma genetika, dan eksperimen pada algoritma *K-means*.

6. Bab 6 Kesimpulan dan Saran
   Bab 6 memuat kesimpulan dari penulis berdasarkan hasil penelitian yang telah dilakukan dan saran untuk peneliti berikutnya agar dapat mengembangkan penelitian ini menjadi lebih baik lagi.

BAB 2

LANDASAN TEORI

Pengelompokan dokumen berkaitan erat dengan dua bidang ilmu dalam informatika. Pe-ngelompokan 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

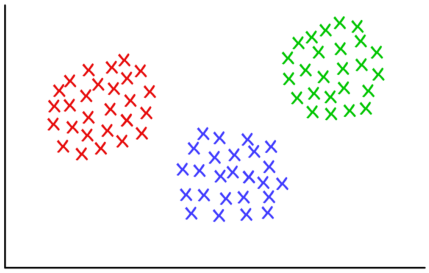
Pengelompokan (*clustering*) merupakan sebuah metode untuk menggabungkan himpunan objek ke dalam kelompok-kelompok sedemikan rupa sehingga objek dalam kelompok (*cluster*) lebih mirip (karena suatu hal) satu sama lain daripada objek di kelompok lain [5]. Pengelompokan seringkali tertukar dengan istilah 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.

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 masukkan yang diberikan oleh pengguna [6]. Ada dua jenis sistem rekomendasi yaitu



Gambar 2.1: Contoh *cluster* hasil pengelompokan

Content-based Recommendation dan *Collaborative Filtering*.

- *Content-based Recommendation*: mempelajari apa yang pengguna sukai lalu mencari objek lain yang mungkin juga disuka 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 disukainya.

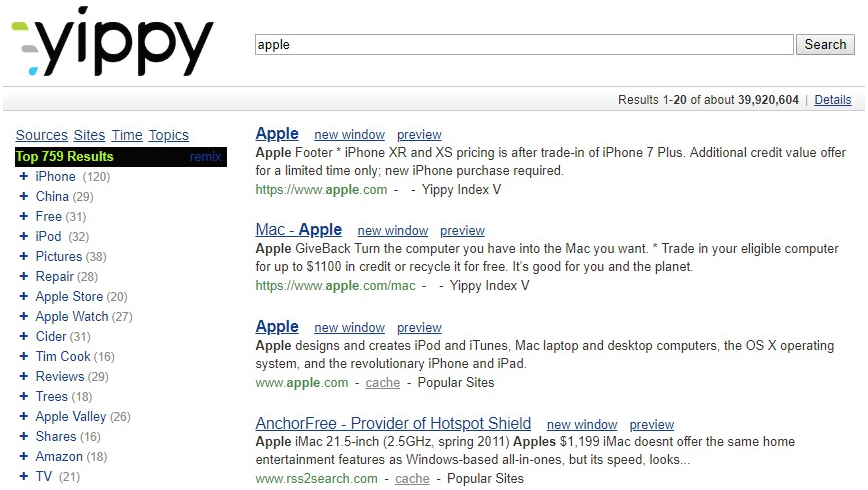
- *Collaborative Filtering*: memberikan usulan berdasarkan apa yang disukai pengguna yang serupa dengan seseorang dengan mengasumsikan jika pengguna serupa menyukai suatu objek, maka orang tersebut akan menyukai objek yang sama. Pengguna serupa didapatkan dengan mencari orang-orang yang berada dalam 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 mengelompokan 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 mengelompokan hasil pencariannya dinamakan dengan *clustering search engine*. Salah satu contoh *clustering search engine adalah* Yippy[1].

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*".

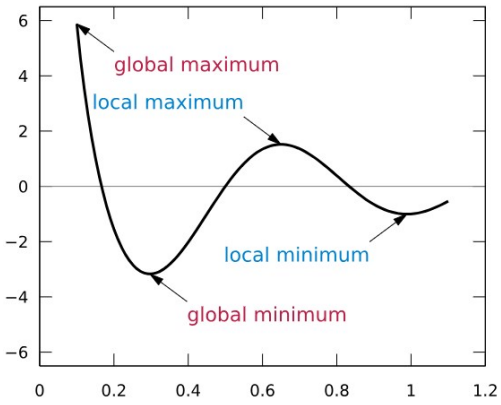
[1] https://yippy.com/



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

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. Persamaaan 2.1 merupakan persamaan untuk menghitung *centroid*

$$\mu_i = \frac{1}{N_i} \sum_{q=1}^{N_i} x_q \tag{2.1}$$

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.

Algoritma 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.

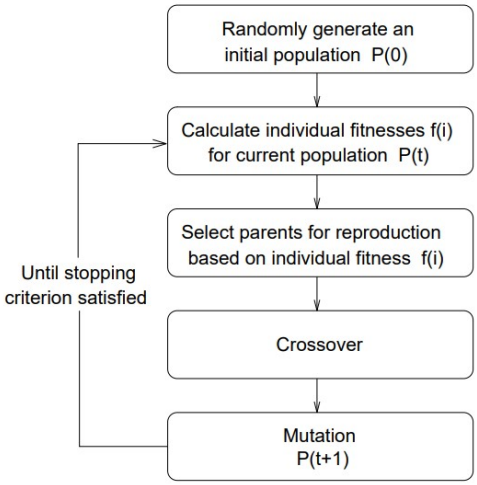
Penjelasan dari Algoritma 1 adalah sebagai berikut:

- Pada baris 4, *centroid* baru akan ditentukan dengan menggunakan Persamaan 2.1.
- Pada baris 5, pengulangan akan berhenti saat *centroid* mengalami pergeseran yang tidak terlalu signifikan (lebih kecil dari suatu nilai ϵ yang merupakan masukan dari pengguna).

2.3 Algoritma Genetika

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

GA merupakan metode penyelesaian masalah yang menggunakan genetika sebagai pemodelannya. Suatu calon solusi dalam GA dimodelkan sebagai suatu individu. Kumpulan individu-individu ini disebut dengan populasi. Setiap individu dalam populasi direpresentasikan dengan kromosom. Kromosom merupakan kumpulan parameter yang membentuk suatu solusi. Parameter-parameter yang menyusun kromosom disebut dengan gen. Setiap kromosom memiliki suatu nilai yang terkait dengan *fitness* dari solusi yang direpresentasikannya. Nilai itu biasanya disebut dengan nilai *fitness*.



Gambar 2.4: Alur algoritma genetika dasar

Secara umum, proses pada GA ditunjukkan dalam Gambar 2.4. Penjelasan dari proses-proses pada GA yang disebutkan dalam Gambar 2.4 adalah sebagai berikut:

1. **Inisialisasi Populasi**
   Inisialisasi populasi merupakan tahap paling awal dari GA. Pada proses ini, akan dibentuk populasi $P(0)$ secara acak.

2. **Perhitungan *Fitness***
   Menghitung nilai *fitness* f(i) dari setiap individu dalam populasi saat ini $P(t)$. Nilai *fitness* ini akan digunakan dalam operasi genetik selanjutnya.

3. **Seleksi**
   Proses seleksi akan terjadi berdasarkan nilai *fitness* f(i) setiap individu. Individu yang lebih *fit* akan memiliki peluang terpilih lebih besar dalam proses seleksi.

4. **Persilangan**
   Individu yang terpilih pada proses seleksi akan disilangkan untuk menghasilkan keturunan. Proses persilangan ini terjadi antara dua induk hasil seleksi.

5. **Mutasi**
   Keturunan yang dihasilkan pada proses persilangan dapat memiliki peluang untuk mengalami mutasi. Mutasi dapat terjadi dengan suatu peluang terjadinya mutasi.

Proses 2 sampai 5 akan diulang terus-menerus sampai ditemukan suatu solusi yang optimal. Berikut merupakan penjelasan lebih lanjut mengenai istilah yang ada pada GA.

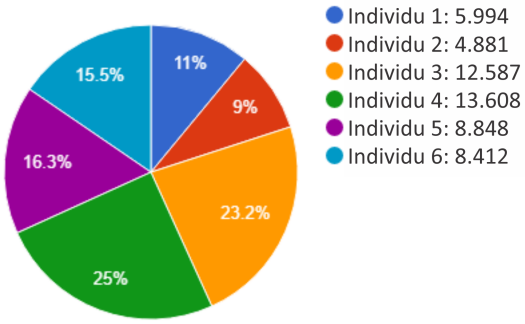
2.3.1 *Fitness*

Fitness dari suatu individu dalam algoritma genetika adalah suatu nilai fungsi objektif untuk fenotipenya [4]. *Fitness* harus bisa memperkirakan seberapa dekat sebuah calon solusi dengan solusi yang optimal. Suatu solusi yang optimal akan memaksimalkan fungsi *fitness*. Perhitungan *fitness* sangatlah tergantung dengan masalah yang ingin diselesaikan. Fungsi *fitness* yang baik harus bisa menentukan solusi mana yang paling baik diantara banyak calon solusi.

2.3.2 Seleksi

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

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. Berdasarkan ilustrasi yang terdapat dalam Gambar 2.5, setiap individu memiliki sebuah bagian pada diagram sesuai dengan nilai *fitness* relatif. Semakin tinggi nilai *fitness*, maka semakin besar bagian yang dialokasikan dan semakin besar kemungkinan individu tersebut akan terpilih dalam proses seleksi.



Gambar 2.5: Ilustrasi *roulette-wheel*

Sebagai penjelasan dari ilustrasi pada Gambar 2.5 mengenai nilai *fitness* relatif adalah sebagai berikut:

- Ada enam individu dalam suatu populasi (Individu 1, Individu 2, dst).
- Setiap individu memililiki nilai *fitness* seperti yang tertera pada gambar (Individu 1 memiliki nilai *fitness* sebesar 5.994, Individu 2 memiliki nilai *fitness* sebesar 4.881, dst).
- Berdasarkan nilai *fitness* tersebut, maka tiap Individu dalam populasi memiliki peluang terpilih yang tercantum dalam persentase pada diagram lingkaran (Individu 1 memiliki peluang terpilih sebesar 11%, Individu 2 memiliki peluang terpilih sebesar 9%, dst). Peluang tersebut dapat dihitung menggunakan persamaan 2.2.

$$P_i = \frac{f_i}{\sum_{j \in Pop} f_j} \tag{2.2}$$

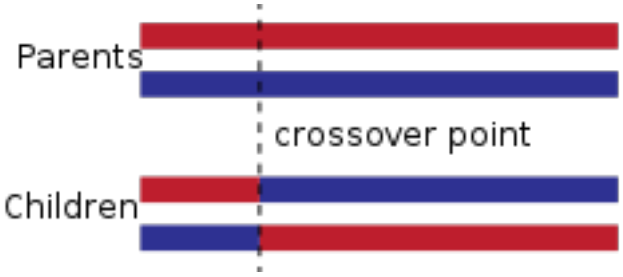
dengan P_i merupakan peluang terpilihnya individu i, f_i merupakan nilai *fitness* dari individu i,Pop merupakan populasi, dan f_j merupakan nilai *fitness* dari individu j.

Proses seleksi dalam GA akan memilih sejumlah induk yang cukup untuk reproduksi dan membentuk generasi selanjutnya. Untuk meningkatkan performa GA, dapat juga 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 hilang dari populasi dalam proses reproduksi.

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*). Gen 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. Oleh karena itu, mutasi juga dapat mencegah GA terjebak di *local optimum* [4]. Mutasi mengubah satu atau beberapa nilai dalam gen. Mutasi terjadi berdasarkan probabilitas mutasi μ_m 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 pengubahan. Apabila terjadi mutasi, satu atau beberapa bagian dari kromosom akan diubah. Mutasi 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 Proses Pencarian Dalam Algoritma Genetika

Proses pencarian dalam GA secara umum dijelaskan dalam Algoritma 2. Proses pencarian ini memanfaatkan operasi genetik yang telah dijelaskan sebelumnya (Subbab 2.3.1 sampai dengan Subbab 2.3.4). Selain itu, terdapat algoritma lain yang menjelaskan masing-masing operasi genetik yang digunakan dalam Algoritma 2. Operasi seleksi yang telah dibahas pada Subbab 2.3.2 dijelaskan

pada Algoritma 4. Operasi persilangan yang telah dibahas pada Subbab 2.3.3 dijelaskan pada Algoritma 3. Operasi mutasi yang telah dibahas pada Subbab 2.3.4 dijelaskan pada Algoritma 5.

Algoritma 2 Algoritma Genetika [9]

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

input: *populasi*, himpunan individu

```
1: solusi ← array yang menyimpan solusi tiap generasi
2: repeat
3:    tambahkan individu dengan nilai fitness tertinggi dari populasi ke solusi
4:    populasi_baru ← himpunan kosong
5:    for i=1 to Size(populasi) do
6:       x ← Seleksi-acak(populasi)
7:       y ← Seleksi-acak(populasi)
8:       anak ← Persilangan(x, y)
9:       rand ← Random(0,1)
10:      if rand ≤ prob_mutasi then
11:         anak ← Mutasi(anak)
12:      end if
13:      tambahkan anak ke populasi_baru
14:   end for
15:   populasi ← 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, solusi pada generasi saat ini akan dicatat ke variabel *solusi*.
- Pada baris 9, variabel *rand* akan berisi bilangan riil antara 0 sampai dengan 1. Bilangan riil ini akan dibangkitkan secara acak dengan distribusi *uniform*.
- Pada baris 16, proses pengulangan akan berhenti saat nilai *fitness* pada N solusi terakhir sudah mengalami konvergensi.
- Pada baris 17, fungsi mengembalikan individu terbaik dalam populasi terakhir GA.

Algoritma 3 Persilangan Algoritma Genetika

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

inputs: x dan y, individu induk

```
1: n ← Length(x)
2: c ← Random(1,N)
3: return Append(Substring(x,1,c), Substring(y,c+1,n))
```

Penjelasan untuk fungsi Persilangan pada Algoritma 3 adalah sebagai berikut:

- Pada baris 2, variabel c akan berisi bilangan bulat antara 1 sampai N. Bilangan bulat ini akan dibangkitkan secara acak 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.

Algoritma 4 Seleksi Algoritma Genetika

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

input: *populasi*, populasi saat ini

```
1: sum ← 0
2: for all individu ∈ populasi do
3:     sum ← sum + Fitness(individu)
4: end for
5: terpilih ← Random(0,1) × sum
6: for all individu ∈ populasi do
7:     terpilih ← terpilih − Fitness(individu)
8:     if terpilih ≤ 0 then
9:         return individu
10:    end if
11: end for
12: return individu dengan urutan terakhir di populasi
```

Penjelasan untuk fungsi Seleksi-acak pada Algoritma 4 adalah sebagai berikut:

- Pada baris 3, fungsi *Fitness* akan mengembalikan nilai *fitness* dari individu yang menjadi parameternya.
- Pada baris 5, variabel *terpilih* akan berisi suatu bilangan 0 sampai *sum*. Hal ini dilakukan dengan cara mengalikan sebuah bilangan riil dengan *sum*. Bilangan riil tersebut merupakan bilangan antara 0 sampai 1 yang dibangkitkan secara acak dengan distribusi *uniform*.

Algoritma 5 Mutasi Algoritma Genetika

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

input: *individu*, individu yang akan dilakukan mutasi

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

Penjelasan untuk fungsi Mutasi pada Algoritma 5 adalah sebagai berikut:

- Pada baris 1, fungsi *Length* akan mengembalikan panjang dari *string* parameterya.
- Pada baris 2, fungsi *Random* akan mengembalikan bilangan bulat acak antara 1 sampai N.

2.3.6 GA dalam Pengelompokan

Algoritma yang umum diterapkan untuk pengelompokan adalah *K-means*. Namun, algoritma *K-means* masih memiliki kekurangan. Salah satu kekurangannya adalah masih dapat terjebak pada *local optimum*. *Local optimum* dapat diatasi oleh GA yang sudah terbukti efektif dalam masalah pencarian dan optimasi [8]. Oleh karena itu, diharapkan pengelompokan berbasis GA dapat menghasilkan solusi yang lebih baik dibandingkan dengan algoritma *K-means*.

2.4 Model Ruang Vektor

Model ruang vektor adalah representasi dari koleksi dokumen sebagai vektor dalam ruang vektor yang umum [10]. 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 2.3.

$$d_i = (w_{1,i}, w_{2,i}, ..., w_{n,i}) \tag{2.3}$$

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 2.4.

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

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.

2.5 Pembobotan *Term* (*Term Weighting*)

Suatu dokumen teks terdiri dari deretan karakter. Sebelum suatu dokumen teks dapat diolah informasinya, maka dokumen tersebut perlu melalui suatu proses yang disebut dengan tokenisasi. Menurut [10], tokenisasi merupakan proses pemotongan suatu dokumen menjadi potongan-potongan (*token*) tertentu. Pada proses yang sama, karakter tertentu akan turut dibuang (tanda baca, spasi, dll). Token adalah urutan karakter dalam dokumen tertentu yang dikelompokkan bersama sebagai unit semantik. Selain token, terdapat juga istilah yang bernama *type* dan *term* . *Type* adalah kelas

dari semua token yang berisi urutan karakter yang sama. *Term* adalah *type* (mungkin dinormalisasi) yang terdapat pada suatu sistem.

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 lebih 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 beberapa cara untuk menghitung bobot suatu *term* . Dua metode yang umum digunakan diantaranya adalah bobot frekuensi (*Frequency weighting*) dan bobot TF-IDF (*TF-IDF weighting*). Bobot TF-IDF merupakan pengembangan dari bobot frekuensi dengan memperhitungkan kemunculan suatu *term* secara global.

2.5.1 Bobot frekuensi

Bobot frekuensi merupakan teknik pembobotan yang sangat sederhana karena bobotnya merupakan jumlah kemunculan *term* tersebut dalam dokumen. *Term* yang sering muncul pada suatu dokumen akan dianggap berkaitan dengan dokumen tersebut. Misalkan suatu dokumen banyak memuat *term* "properti", maka dokumen tersebut dianggap merupakan suatu dokumen yang membahas masalah "properti". Bobot frekuensi dapat digambarkan dengan Persamaan 2.5

$$w_i = tf_i \tag{2.5}$$

dengan w_i merupakan bobot *term* ke-i dan tf_i merupakan frekuensi kemunculan *term* ke-i pada dokumen. Sebagai ilustrasi, akan digunakan empat buah dokumen (masing-masing terdiri dari satu kalimat) yang berasal dari contoh pada [10] sebagai berikut:

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

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

Term "*in*" dalam tabel pada d1 dan d4 bernilai 0 karena *term* "*in*" tidak muncul pada d1 dan d4. Sedangkan pada d2, *term* "*in*" muncul satu kali sehingga bernilai 1 pada tabel. Begitu juga pada d3, *term* "*in*" muncul dua kali sehingga bernilai 2 pada tabel.

2.5.2 Bobot TF-IDF

Selain menggunakan bobot frekuensi, ada teknik pembobotan lain yang disebut dengan TF-IDF (*Term Frequency-Inverse Document Frequency*). Teknik pembobotan ini merupakan pengembangan

Term	d1	d2	d3	d4
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*

dari pembobotan frekuensi. TF-IDF merupakang gabungan dari *term frequency* (tf) dengan *inverse document frequency* (idf) untuk menghasilkan suatu bobot komposit untuk setiap *term* dalam setiap dokumen [10].

Term frequency atau biasa dilambangkan sebagai $\mathrm{tf}_{t,d}$ untuk *term* t pada dokumen d. Sama seperti yang telah dijelaskan pada Subbab 2.5.1, TF merupakan banyaknya kemunculan *term* pada suatu dokumen. Namun pada TF-IDF, umumnya digunakan bobot frekuensi yang telah dinormalisasi dengan jumlah kemunculan semua term pada suatu dokumen. TF yang telah dinormalisasi dapat dihitung menggunakan persamaan 2.6.

$$\mathrm{tf}_{t,d} = \frac{\mathrm{f}_{t,d}}{\sum_{t' \in d} \mathrm{f}_{t',d}} \tag{2.6}$$

dengan $\mathrm{tf}_{t,d}$ merupakan TF *term* t pada dokumen d, $\mathrm{f}_{t,d}$ merupakan banyaknya kemunculan *term* t dalam dokumen d, $t' \in d$ merupakan seluruh *term* t' yang merupakan anggota dari dokumen d, dan $\mathrm{f}_{t',d}$ merupakan banyaknya kemunculan *term* t' pada dokumen d.

Terdapat suatu masalah pada pengukuran menggunakan TF yaitu semua *term* dianggap sama penting. Pada kenyatannya, tidak semua *term* itu sama penting karena *term* yang sangat umum akan memiliki frekuensi yang sangat besar pada banyak dokumen. Seperti yang sudah dijelaskan pada Subbab 2.5.1, suatu *term* yang sering muncul akan dianggap berkaitan dengan suatu dokumen. Namun hal itu ternyata belum tentu benar karena banyak *term* yang sering muncul padahal *term* tersebut sebenarnya tidak memiliki nilai informasi seperti *term* yang merupakan kata penghubung. Oleh karena itu, diperlukan suatu mekanisme untuk mengurangi efek dari suatu *term* yang sering muncul di banyak dokumen. Salah satu cara yang digunakan untuk menangani hal tersebut adalah dengan menggunakan *inverse document frequency* (IDF). IDF ini dapat dihitung dengan menggunakan persamaan 2.7.

$$\mathrm{idf}_t = \log \frac{N}{\mathrm{df}_t} \tag{2.7}$$

dengan idf_t merupakan idf dari *term* t, N merupakan banyaknya anggota himpunan dokumen, dan df_t merupakan *document frequency* dari *term* t. *Document frequency* adalah banyaknya dokumen pada himpunan dokumen yang memuat *term* t.

Bobot TF-IDF menggabungkan teknik pembobotan TF dengan IDF. Nilai dari TF digabungkan dengan nilai dari IDF dengan cara mengalikan keduanya. Metode TF-IDF ini sangat populer

digunakan oleh sistem rekomendasi berbasis teks [11]. Pembobotan menggunakan TF-IDF dapat dihitung menggunakan Persamaan 2.8.

$$\text{tf-idf}_{t,d} = \text{tf}_{t,d} \times \text{idf}_t \tag{2.8}$$

Berdasarkan rumus tersebut, maka dapat ditarik dua kesimpulan yaitu:

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

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 *Intracluster* untuk Mengukur Kinerja Metode *Clustering*

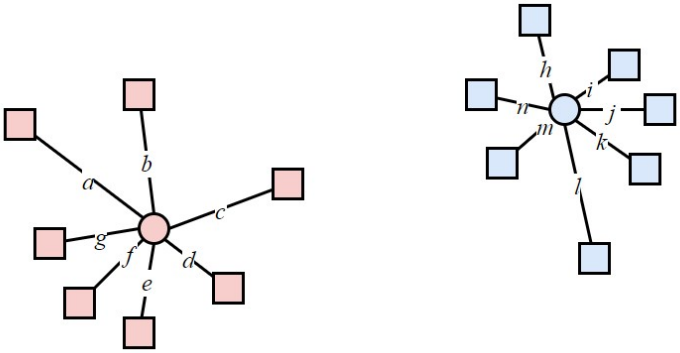
Untuk mengukur performa dari suatu algoritma, perlu dilakukan pengujian. Pengujian ini dapat menentukan apakah algoritma yang digunakan sudah cukup baik dalam menyelesaikan masalah yang dibuat. Dalam pengelompokan, ada beberapa cara untuk mengukur apakah objek-objek sudah berhasil dikelompokan secara baik atau tidak. Cara yang pertama adalah dengan mengukur jarak antara tiap objek ke titik pusat *cluster* (*centroid*) atau biasa disebut jarak *intracluster*. 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 *intracluster*. Jarak *intracluster* dapat diukur dengan cara menjumlahkan jarak setiap objek ke masing-masing titik pusatnya. Cara untuk menghitung jarak *intracluster* ditunjukkan dalam Persamaan 2.9.

$$\begin{aligned} M &= \sum_{i=1}^{K} M_i, \\ M_i &= \sum_{x_j \in C_i} \| x_j - z_i \| \end{aligned} \tag{2.9}$$

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. Untuk memperjelas perhitungan, Gambar 2.7 akan digunakan sebagai ilustrasi dari jarak *intracluster*.

Pada Gambar 2.7, terdapat 14 objek yang telah dibagi ke dalam dua *cluster* berwarna merah dan biru. Persegi pada Gambar **??** mengilustrasikan objek yang akan dikelompokkan. Lingkaran mengilustrasikan titik pusat *cluster* (*centroid*). Garis yang menghubungkan objek dan *centroid* menggambarkan jarak antara objek dengan *centroid* dan ditandai dengan label antara huruf *a*



Gambar 2.7: *Ilustrasi untuk jarak intracluster*

sampai dengan n. Persegi berwarna merah merupakan anggota dari *cluster* 1, sedangkan persegi berwarna biru merupakan anggota dari *cluster* 2. Berdasarkan Persamaan 2.9, jarak *intracluster* dari *cluster* 1 (M_1) didapat dengan menjumlahkan a, b, c, d, e, f, dan g. Sedangkan, jarak *intracluster* dari *cluster* 2 (M_2) didapat dengan menjumlahkan h, i, j, k, l, m, dan n. Jarak *intracluster* total didapatkan dengan menjumlahkan jarak *intracluster* miliki *cluster* 1 dan *cluster* 2 ($a + b + ... + n$).

Semakin kecil nilai M, maka pengelompokan dianggap semakin baik. Ini berarti setiap objek dalam suatu*cluster* mirip satu sama lain. Metrik ini dapat mengukur seberapa baik objek sudah dikelompokan dengan mempertimbangkan kedekatan setiap objek ke titik pusatnya.

2.7 Evaluasi Hasil Pengelompokan Menggunakan *Purity*

Selain metode perhitungan jarak *intracluster* yang telah dijelaskan pada Subbab 2.6, Diperlukan cara eksternal yang dapat mengukur kualitas dari pengelompokan. Pada bagian ini akan dijelaskan salah satu metode eksternal yaitu *purity*. *Purity* adalah persentase objek yang berhasil dikelompokan dengan benar [10]. Untuk menghitung *purity*, diperlukan suatu *dataset* yang sudah diberi label. Label ini kemudian akan digunakan untuk menghitung akurasi dari pengelompokan itu sendiri. Cara untuk menghitung *purity* ditunjukkan dalam Persamaan 2.10.

$$purity = \frac{1}{N} \sum_{i=1}^{k} max_j |c_i \cap t_j| \tag{2.10}$$

dengan N merupakan banyaknya objek, k merupakan banyakya *cluster*, c_i merupakan *cluster* ke-i, dan t_j adalah label j. Rentang dari nilai *purity* adalah antara 0 sampai dengan 1. *Purity* bernilai 1 apabila tidak ada objek dengan 2 atau lebih label berbeda dalam satu kelompok yang sama. Apabila nilai *purity*semakin mendekati 0, maka hasil pengelompokan semakin tidak murni (terdapat banyak objek dengan label berbeda dalam satu kelompok).

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.

Term	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 2.8 pada Subbab 2.5.2. Sama dengan perhitungan bobot frekuensi, contoh yang berasal

dari Subbab 3.3 akan digunakan sebagai ilustrasi perhitungan TF-IDF. Sesuai dengan apa yang telah dijelaskan pada Subbab 2.5.2, perhitungan dari TF-IDF akan dibagi menjadi dua tahap yaitu perhitungan TF dan perhitungan IDF.

Term "properti" akan digunakan dalam contoh perhitungan TF. Berdasarkan Persamaan 2.6 pada Subbab 2.5.2, maka perhitungan TF dari *term* "properti" ditunjukkan pada Persamaan 3.1.

$$\begin{aligned} \mathrm{tf}_{t,d} &= \frac{\mathrm{f}_{t,d}}{\sum_{t' \in d} \mathrm{f}_{t',d}} \\ \mathrm{tf}_{\text{"}properti\text{"},1} &= \frac{1}{8} = 0.125 \end{aligned} \tag{3.1}$$

Penjelasan dari Persamaan 3.1 adalah sebagai berikut. $\mathrm{f}_{\text{"}properti\text{"},1}$ bernilai 1 karena *term* "properti" hanya muncul 1 kali pada dokumen 1. Dokumen 1 terdiri dari 8 *term* sehingga $\mathrm{tf}_{\text{"}properti\text{"},1}$ bernilai 0.125. Berdasarkan Persamaan 3.1, maka hasil perhitungan TF dari ketiga dokumen tersebut ditunjukkan pada Tabel 3.2.

Term	Bobot		
	Dokumen 1	Dokumen 2	Dokumen 3
penjualan	0.1250	0.1667	0.1429
properti	0.1250	0	0
di	0.2500	0.1667	0
indonesia	0.1250	0.1667	0
meningkat	0.1250	0.1667	0
bulan	0.1250	0	0.2857
februari	0.1250	0	0.1429
asuransi	0	0.1667	0
kendaraan	0	0.1667	0.1429
merupakan	0	0	0.1429
puncak	0	0	0.1429

Tabel 3.2: Hasil perhitungan TF

Untuk perhitungan IDF akan digunakan Persamaan 2.7 pada Subbab 2.5.2. Perhitungan IDF untuk *term* "properti" ditunjukkan oleh Persamaan 3.2.

$$\begin{aligned} \mathrm{idf}_t &= \log \frac{N}{\mathrm{df}_t} \\ \mathrm{idf}_{\text{"}properti\text{"}} &= \log \frac{3}{1} = 0.4771 \end{aligned} \tag{3.2}$$

Penjelasan dari Persamaan 3.2 adalah sebagai berikut. N bernilai 3 karena banyaknya dokumen dalam seluruh koleksi dokumen adalah 3. df_t bernilai 1 karena hanya ada 1 dokumen yang memuat *term* "properti" yaitu dokumen 1. Hasil dari perhitungan IDF untuk setiap *term* ditunjukkan oleh Tabel 3.3.

Term	IDF
penjualan	0
properti	0.4771
di	0.1761
indonesia	0.1761
meningkat	0.1761
bulan	0.1761
februari	0.1761
asuransi	0.4771
kendaraan	0.1761
merupakan	0.4771
puncak	0.4771

Tabel 3.3: Hasil perhitungan IDF

Term penjualan pada Tabel 3.3 bernilai 0. *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.

Hasil dari perhitungan TF dan IDF digabung dengan cara dikalikan. Perhitungan TF-IDF untuk *term* "properti" pada dokumen 1 ditunjukkan dalam Persamaan 3.3.

$$\begin{aligned} \text{tf-idf}_{t,d} &= \text{tf}_{t,d} \times \text{idf}_t \\ \text{tf-idf}_{\text{"}properti\text{"},1} &= 0.125 \times 0.4771 = 0.0075 \end{aligned} \tag{3.3}$$

Nilai yang berasal dari Persamaan 3.1 dan Persamaan 3.2 dikalikan sehingga didapatkan hasil sesuai dengan Persamaan 3.3. Hasil perhitungan TF-IDF untuk seluruh *term* dalam ketiga dokumen ditunjukkan dalam Tabel 3.4.

Term	Bobot		
	Dokumen 1	Dokumen 2	Dokumen 3
penjualan	0	0	0
properti	0.0075	0	0
di	0.0055	0.0049	0
indonesia	0.0028	0.0049	0
meningkat	0.0028	0.0049	0
bulan	0.0028	0	0.0072
februari	0.0028	0	0.0036
asuransi	0	0.0133	0
kendaraan	0	0.0049	0.0036
merupakan	0	0	0.0097
puncak	0	0	0.0097

Tabel 3.4: Hasil perhitungan bobot TF-IDF

3.4 Representasi Kromosom

Individu dalam pengelompokan merupakan salah satu cara pengelompokan. Dalam penelitian ini, individu direpresentasikan oleh sebuah kromosom. Kromosom tersusun dari gen-gen yang

merupakan *centroid* dari K buah *cluster* yang akan dibentuk. *Centroid* disimpan dalam bentuk vektor yang terdiri dari N buah bilangan riil. N merupakan dimensi dari vektor tersebut yang merupakan banyaknya *term* berbeda yang terdapat pada seluruh koleksi dokumen. Oleh karena itu, setiap kromosom akan memiliki $N \times K$ buah gen. Secara umum representasi *centroid* ke dalam kromosom ditunjukkan pada Gambar 3.1.

$$\underbrace{w_{1,1}, w_{1,2}, ..., w_{1,n},}_{C_1} \quad \underbrace{w_{2,1}, w_{2,2}, ..., w_{2,n}}_{C_2} \quad ... \quad \underbrace{w_{k,1}, w_{k,2}, ..., w_{k,n}}_{C_k}$$

Gambar 3.1: Formula representasi kromosom

Berdasarkan Gambar 3.1, N gen pertama merepresentasikan N dimensi dari *centroid* pertama C_1 $(w_{1,1}, w_{1,2}, ..., w_{1,n})$, N gen selanjutnya merepresentasikan N dimensi dari *centroid* kedua C_2 $(w_{2,1}, w_{2,2}, ..., w_{2,n})$, dan seterusnya sampai *centroid* C_k $(w_{k,1}, w_{k,2}, ..., w_{k,n})$. 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 Gambar 3.2.

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

Gambar 3.2: Contoh representasi *centroid* ke dalam kromosom

3.5 Fungsi *Fitness*

Seperti yang telah dijelaskan pada Subbab 3.4, kromosom akan tersusun atas *centroid* yang berbentuk vektor. Oleh karena itu, perhitungan *fitness* akan mengalami beberapa penyesuaian. Perhitungan *fitness* dalam penelitian ini terdiri dari tiga tahap. Pada tahap pertama, terjadi pembentukan *cluster* berdasarkan titik pusat yang terkandung dalam kromosom. Hal ini dilakukan dengan menetapkan setiap dokumen $x_i, i = 1, 2, ..., n$ ke dalam sebuah *cluster* C_j dengan *centroid* z_j sehingga memenuhi Persamaan 3.4.

$$\|x_i - z_j\| < \|x_i - z_p\|, p = 1, 2, ..., K, \text{ dan } p \neq j. \tag{3.4}$$

Persamaan 3.4 menjelaskan bahwa akan dicari suatu *cluster* C_j yang paling dekat dengan dokumen x_i. Hal ini dilakukan dengan cara membandingkan jarak antara dokumen x_i dengan seluruh *centroid* dari K buah *cluster*.

Setelah proses pengelompokan selesai, maka akan dilanjutkan dengan tahap kedua yaitu mengganti *centroid* terkandung dalam kromosom dengan *centroid* baru. *Centroid* baru ini ditentukan dengan cara menghitung rata-rata vektor dari tiap *cluster*. Untuk *cluster* C_i, *centroid* baru z_i^* dapat dihitung menggunakan Persamaan 3.5.

$$z_i^* = \frac{1}{n_i} \sum_{x_j \in C_i} x_j, i = 1, 2, ..., K. \tag{3.5}$$

dengan z_i^* merupakan *centroid* dari *cluster* i, n_i merupakan jumlah anggota *cluster* i, dan x_j merupakan dokumen j yang merupakan anggota dari *cluster* i. z_i^* ini akan menggantikan z_i sebelumnya di kromosom.

Tahap terakhir dari perhitungan *fitness* adalah menghitung nilai *fitness* itu sendiri. Pada penelitian ini, nilai *fitness* akan dihitung menggunakan *cosine similarity* seperti yang sudah dijelaskan pada Subbab 2.4. Sesuai dengan Persamaan 2.9 pada Subbab 2.6, perhitungan *fitness* sebagai metrik yang menggambarkan seberapa baiknya suatu calon solusi ditunjukkan dalam Persamaan 3.6 dengan beberapa penyesuaian.

$$\begin{aligned} \mathrm{f} &= \sum_{i=1}^{K} \mathrm{f}_i, \\ \mathrm{f}_i &= \sum_{x_j \in C_i} \frac{x_j \cdot z_i}{\parallel x_j \parallel \times \parallel z_i \parallel} \end{aligned} \tag{3.6}$$

Penjelasan dari Persamaan 3.6 adalah sebagai berikut. Nilai *fitness* f didapatkan dengan menjumlahkan nilai *fitness* f_i setiap *centroid* $i = 1, 2, ..., K$. Nilai *fitness* f_i tiap *centroid* didapatkan dengan menjumlahkan jarak setiap dokumen ke *centroid* masing-masing (Subbab 2.6). Perhitungan jarak antara dokumen dengan *centroid* akan dihitung dengan menggunakan *cosine similarity* sehingga persamaan untuk menghitung *cosine similarity* (Persamaan 2.4 pada Subbab 2.4) dimasukkan ke dalam Persamaan 3.6. Semakin besar nilai *fitness* f, maka kromosom tersebut semakin mendekati solusi yang optimal.

3.6 Operasi Genetik Dalam Pengelompokan Dokumen

Seperti yang telah di bahas pada Subbab 2.3, ada beberapa operator genetik yang digunakan dalam algoritma genetika. Beberapa operasi genetik yang akan digunakan dalam penelitian ini di antaranya adalah inisialisasi populasi, seleksi, persilangan, dan mutasi. Sama seperti kromosom, operator genetik ini juga perlu dimodifikasi sedemikian rupa sehingga dapat digunakan untuk proses pengelompokan dokumen. Berikut merupakan pembahasan lebih detil mengenai setiap operator genetik yang akan digunakan.

3.6.1 Inisialisasi Populasi

Proses pengelompokan dengan menggunakan algoritma genetika akan dimulai dengan inisialisasi populasi awal. Seperti yang sudah dijelaskan pada Subbab 3.4, kromosom dari masing-masing individu akan terdiri dari K buah *centroid* dalam bentuk vektor. Populasi awal akan dibangkitkan dengan cara memilih K dokumen secara acak yang akan dijadikan *centroid* mula-mula. Kemudian proses tersebut akan dilakukan sebanyak P kali dengan P adalah banyaknya individu dalam populasi.

3.6.2 Seleksi

Proses seleksi akan dilakukan setiap iterasi (setiap generasi) untuk memilih calon induk dari generasi selanjutnya. Pada penelitian ini akan digunakan teknik seleksi *roulette-wheel selection* seperti yang telah dijelaskan pada Subbab 2.3.2. *Roulette-wheel selection* digunakan secara langsung pada penelitian ini dan tidak dimodifikasi. Individu dengan nilai *fitness* lebih tinggi akan memiliki probabilitas lebih tinggi untuk terpilih menjadi induk dari generasi selanjutnya dan akan masuk ke dalam proses persilangan. Namun karena masih ada peluang individu dengan nilai *fitness*

tertinggi tidak terpilih, maka dalam penelitian ini juga akan digunakan teknik elitisme [12]. Dengan digunakannya elitisme, maka individu dengan nilai *fitness* terbaik akan disalin dan langsung menjadi anggota dari generasi berikutnya. Hal ini dilakukan untuk menjamin individu terbaik tidak hilang akibat tidak terpilih oleh *roulette-wheel selection*.

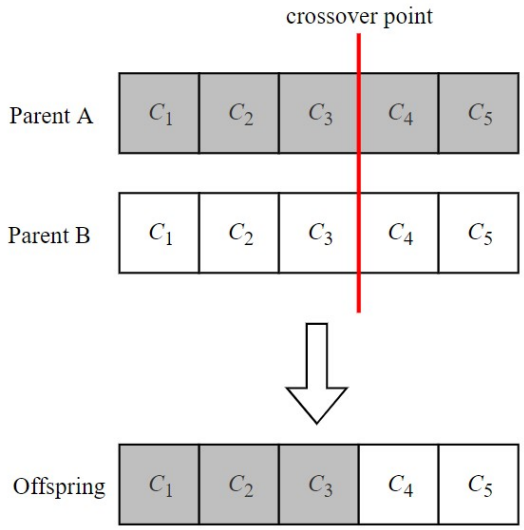
3.6.3 Persilangan

Dua kromosom yang terpilih dalam proeses seleksi akan disilangkan untuk menghasilkan keturunan yang akan dimasukkan ke dalam generasi selanjutnya. Dalam penelitian ini akan digunakan teknik persilangan dengan satu titik potong (*single-point crossover*). Teknik *single-point crossover* yang digunakan dalam penelitian ini sedikit berbeda karena perlu disesuaikan dengan kebutuhan pengelompokan dokumen. Titik potong tidak ditentukan pada tingkat gen, namun ditentukan pada tingkat *centroid*. Apabila diketahui terdapat kromosom seperti pada Gambar 3.3, maka titik potong yang dipilih bukan berupa gen ($w_{k,n}$) melainkan *centroid* (C_k).

C_1	C_2	C_3	C_4	C_5
$w_{1,1}, w_{1,2}, ..., w_{1,n},$	$w_{2,1}, w_{2,2}, ..., w_{2,n},$	$w_{3,1}, w_{3,2}, ..., w_{3,n},$	$w_{4,1}, w_{4,2}, ..., w_{4,n},$	$w_{5,1}, w_{5,2}, ..., w_{5,n}$

Gambar 3.3: Ilustrasi kromosom untuk persilangan

Sebagai contoh, diketahui dua buah kromosom yang akan mengalami persilangan adalah kromosom A dengan kromosom B seperti pada Gambar 3.4. Setelah itu akan ditentukan suatu titik potong (*crossover point*) secara acak dan ditandai dengan garis tegak berwarna merah pada Gambar 3.4. Proses selanjutnya adalah pembentukan individu keturunan. Proses ini dilakukan dengan cara menggabungkan seluruh gen induk A yang berada di sebelah kiri garis merah dengan seluruh gen induk B yang berada di sebelah kanan garis merah. Keturunan yang dihasilkan dari proses ini hanya satu individu seperti yang dapat dilihat pada Gambar 3.4.



Gambar 3.4: Ilustrasi persilangan

3.6.4 Mutasi

Setiap individu keturunan yang berasal dari proses persilangan akan memiliki peluang untuk mengalami mutasi. Seperti yang telah dijelaskan pada Subbab 2.3.4, mutasi dapat terjadi berdasarkan peluang mutasi μ_m. 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 Gambar 3.5a, akan ditentukan satu dari enam gen yang akan mengalami mutasi. Misalkan dalam contoh ini gen kedua yang mengalami mutasi (Gambar 3.5b). 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 Gambar 3.5c.

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

(a) Kromosom sebelum mutasi

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

(b) Pemilihan gen yang akan dimutasi

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

(c) Kromosom hasil mutasi

Gambar 3.5: Ilustrasi mutasi kromosom

BAB 4

PERANCANGAN

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

4.1 Kebutuhan Masukan dan Keluaran

Seluruh kebutuhan masukan dari perangkat lunak ini akan diakomodasi oleh antarmuka pengguna (Graphical User Interface). Rincian masukan melalui antarmuka pengguna akan dibahas lebih lanjut pada Subbab 4.3. Keluaran dari perangkat lunak ini adalah sebuah *file* berformat *CSV* sehingga dapat dibuka dan diolah lebih lanjut dengan perangkat pengolah *spreadsheet* seperti Microsoft Excel. *File* ini berisi laporan mengenai parameter masukan, hasil pengelompokan, nilai fitness, dan waktu yang dibutuhkan untuk melakukan seluruh proses tersebut. Contoh *file* tersebut dapat dilihat pada Tabel 4.1.

File ini dinamai berdasarkan algoritma dan waktu penulisan dengan format "algoritma-YYYY.MM.DD hh_mm_ss.csv". Sebagai contoh apabila pengelompokan menggunakan algoritma genetika dan dilakukan pada tanggal 30 Januari 2019 pukul 08:15:30 maka nama dari *file* yang dihasilkan adalah "GA-2019.01.30 08_15_30.csv". Apabila algoritma yang digunakan adalah *K-means* maka nama *file* yang dihasilkan adalah "KMeans-2019.01.30 08_15_30.csv".

Direktori Dokumen: D:\dataset\bbc

Parameter:

Banyaknya Cluster	5
Banyaknya Populasi	100
Metode Pembobotan	TF-IDF
Probabilitas Mutasi	0.05
Maksimum Iterasi	100
Individu Elitisme	1
Banyaknya Generasi Konvergen	3
Batas Konvergen	1.00E-05

Hasil:

Waktu	20 menit 6 detik
Intracluster	565.9675341
Banyak Iterasi	4

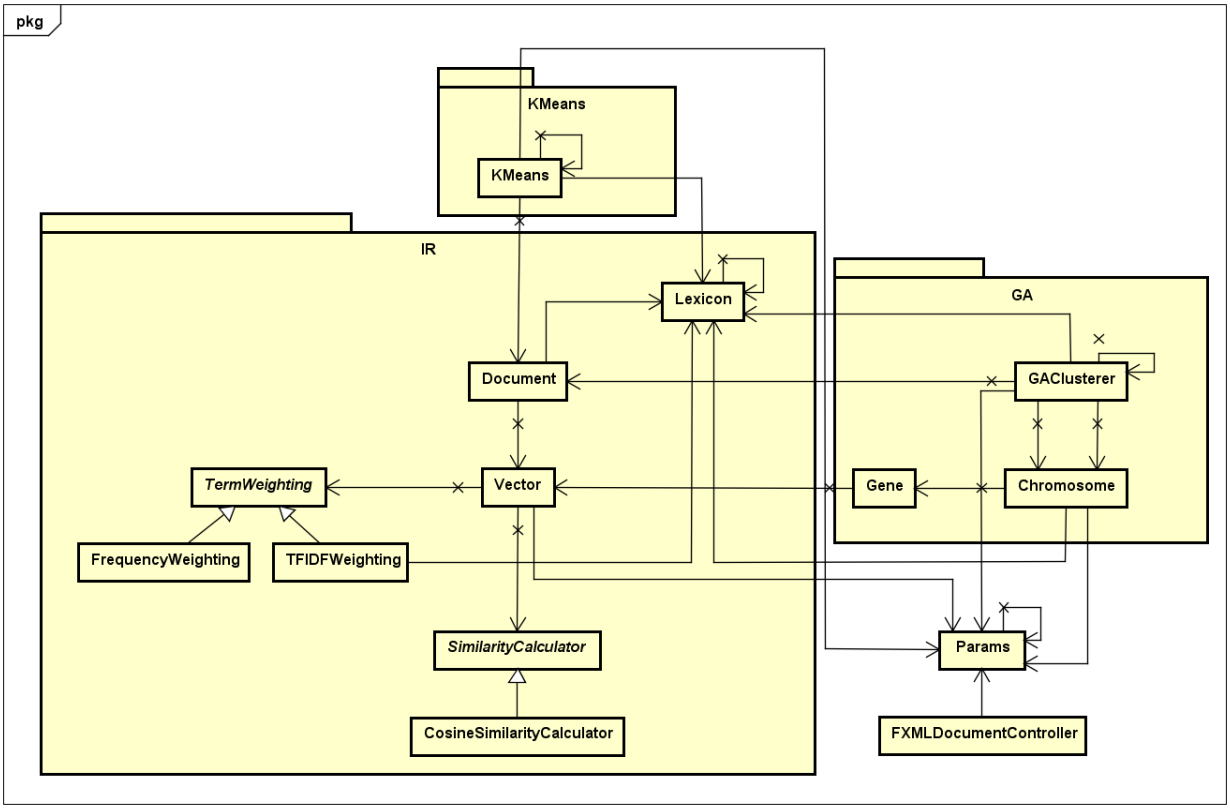
Hasil Clustering:

C1	C2	C3	C4	C5
tech\020.txt	entertainment\267.txt	tech\207.txt	sport\418.txt	sport\262.txt
tech\128.txt	entertainment\279.txt	tech\234.txt	sport\134.txt	sport\391.txt
tech\012.txt	business\220.txt	tech\229.txt	sport\394.txt	sport\389.txt
tech\014.txt	entertainment\342.txt	tech\005.txt	sport\379.txt	sport\243.txt
tech\334.txt	sport\471.txt	tech\359.txt	sport\305.txt	sport\206.txt
...	...	...	...	...

Tabel 4.1: Contoh *file CSV* hasil pengelompokan dokumen

4.2 Rancangan Kelas

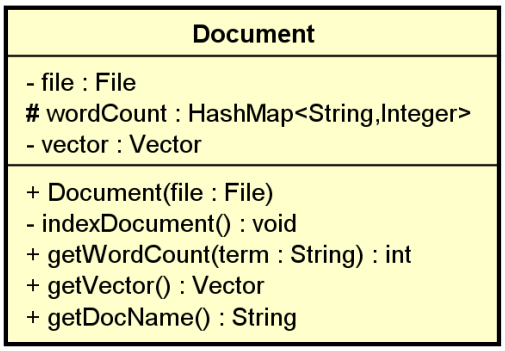
Berdasarkan hasil analisis dari masalah yang dihadapi, dibentuklah diagram kelas pada Gambar 4.1 sebagai gambaran dari perangkat lunak yang akan dibuat.



Gambar 4.1: Diagram kelas

Gambar 4.1 merupakan diagram kelas secara umum yang tidak memuat atribut dan *method* dari setiap kelas. Diagram kelas ini sengaja dibuat agar hubungan antar kelas dapat dengan lebih mudah dilihat. Penjelasan dari setiap kelas dalam Gambar 4.1 adalah sebagai berikut.

4.2.1 *Document*



Gambar 4.2: Kelas *Document*

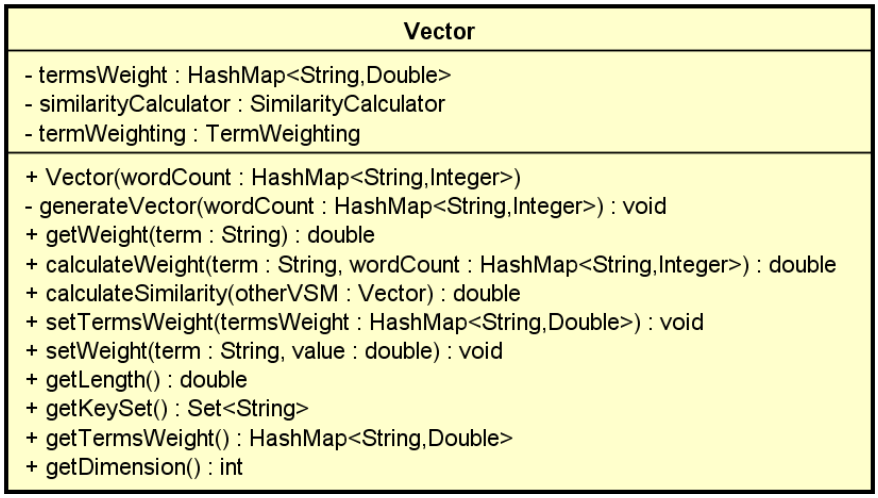
Kelas ini merupakan representasi dari dokumen yang akan diproses dalam pengelompokan. Kelas ini berfungsi untuk menyimpan informasi yang dibutuhkan dari sebuah dokumen selama proses pengelompokan. Atribut yang dimiliki oleh kelas *Document* adalah:

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

Method yang terdapat dalam kelas ini adalah:

- *Document*: *method* ini 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*: *method* yang berfungsi untuk mengembalikan banyaknya *term* muncul dalam dokumen.
- *getVector*: *method* ini merupakan *getter* dari atribut *vector*.
- *getDocName*: *method* ini mengembalikan nama *file* dari dokumen.

4.2.2 *Vector*



Gambar 4.3: Kelas *Vector*

Kelas ini merupakan kelas yang merepresentasikan sebuah vektor. Kelas ini memiliki fungsi dan atribut yang berfungsi untuk menunjang seluruh aktivitas yang melibatkan suatu vektor. Atribut yang dimiliki kelas ini adalah:

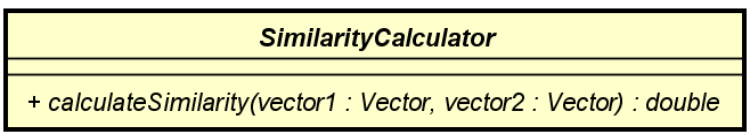
- *termsWeight*: atribut ini bertipe *Hashmap* dengan *key* berupa *String* dan *value* berupa *Double*. Atribut ini menyimpan pasangan *term* dan bobotnya sesuai dengan metode pembobotan.

- *similarityCalculator*: atribut ini bertipe *SimilarityCalculator* dan merupakan objek yang akan digunakan untuk menghitung kemiripan antar vektor.
- *termWeighting*: atribut ini bertipe *TermWeighting* dan merupakan objek yang akan digunakan untuk menghitung bobot dari setiap dimensi dalam suatu vektor.

Method yang terdapat dalam kelas ini adalah:

- *Vector*: *method* ini merupakan constructor dengan sebuah parameter yaitu *wordCount* bertipe *Hashmap<String,Integer>* yang merupakan pasangan kata dan banyak kemunculannya dalam dokumen.
- *generateVector*: *method* ini merupakan *method* dengan sebuah parameter *Hashmap<String,Integer>* untuk mengisi atribut *termsWeight* dengan bobot tiap *term* .
- *getWeight*: *method* ini berfungsi untuk mengembalikan bobot dari *term* .
- *calculateWeight*: *method* ini membutuhkan dua buah parameter yaitu sebuah *term* bertipe *String* dan sebuah *Hashmap<String,Integer>*. *Method* ini berfungsi untuk menghitung bobot dari *term* berdasarkan frekuensi yang terdapat pada *HashMap*.
- *calculateSimilarity*: *method* ini berfungsi untuk menghitung kemiripan (*similarity*) antara vektor ini dengan *otherVector* menggunakan metode yang dipilih oleh pengguna.
- *setTermsWeight*: *method* ini merupakan *setter* dari atribut *termsWeight*.
- *setWeight*: *method* ini berfungsi untuk mengubah bobot *term* menjadi *value*.
- *getLength*: *method* ini berfungsi untuk mendapatkan panjang dari vektor.
- *getKeyset*: *method* ini berfungsi untuk mendapatkan himpunan *term* yang ada pada vektor ini.
- *getTermsWeight*: *method* ini merupakan *getter* dari atribut *termsWeight*.
- *getDimension*: *method* ini berfungsi untuk mendapatkan besarnya dimensi dari vektor ini.

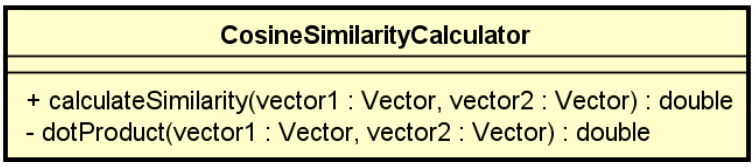
4.2.3 *SimilarityCalculator*



Gambar 4.4: Kelas *SimilarityCalculator*

Kelas ini merupakan kelas abstrak yang berfungsi untuk menghitung kemiripan antara dua buah objek bertipe *Vector*. Kelas ini tidak memiliki atribut dan hanya memiliki sebuah *method* abstrak yaitu *calculateSimilarity* yang memiliki dua buah parameter *vector1* dan *vector2*. Hasil yang dikembalikan oleh *method* ini adalah kemiripan dari kedua vektor tersebut sesuai dengan metode perhitungan jaraknya.

4.2.4 *CosineSimilarityCalculator*

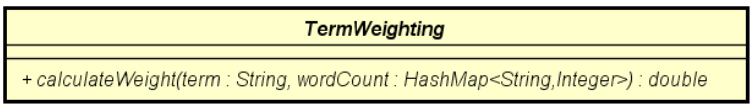


Gambar 4.5: Kelas *CosineSimilarityCalculator*

Kelas ini mengimplementasikan kelas abstrak *SimilarityCalculator*. Kelas ini memiliki satu *method* tambahan selain melakukan *override* pada *method calculateSimilarity*. *Method* yang ada pada kelas ini adalah:

- *calculateDistance*: *method* ini merupakan *method* yang diturunkan dari kelas *SimilarityCalculator*. *Method* ini mengembalikan *similarity* dari *vector1* dan *vector2* yang dihitung menggunakan persamaan cosinus (Persamaan 2.4).
- *dotProduct*: *method* ini berfungsi untuk menghitung hasil perkalian titik (*dot product*) antara *vector1* dan *vector2*.

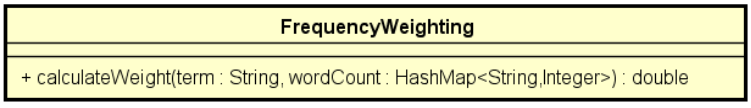
4.2.5 *TermWeighting*



Gambar 4.6: Kelas *TermWeighting*

Kelas ini merupakan kelas abstrak yang merepresentasikan metode perhitungan bobot dalam suatu vektor. Kelas ini hanya memiliki satu buah *method* yaitu *calculateWeight*. *Method* ini berfungsi untuk menghitung bobot dari *term* berdasarkan metode pembobotan yang dipilih oleh pengguna.

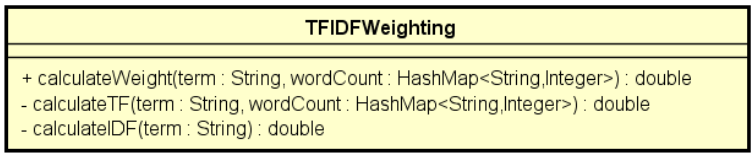
4.2.6 *FrequencyWeighting*



Gambar 4.7: Kelas *FrequencyWeighting*

Kelas ini mengimplementasikan kelas abstrak *TermWeighting*. Kelas ini hanya memiliki satu *method* yang diturunkan langsung dari kelas *TermWeighting* yaitu *method calculateWeight*. *Method* ini berfungsi untuk menghitung bobot dari *term* menggunakan bobot frekuensi seperti yang telah dijelaskan pada Subbab 2.5.1.

4.2.7 *TFIDFWeighting*

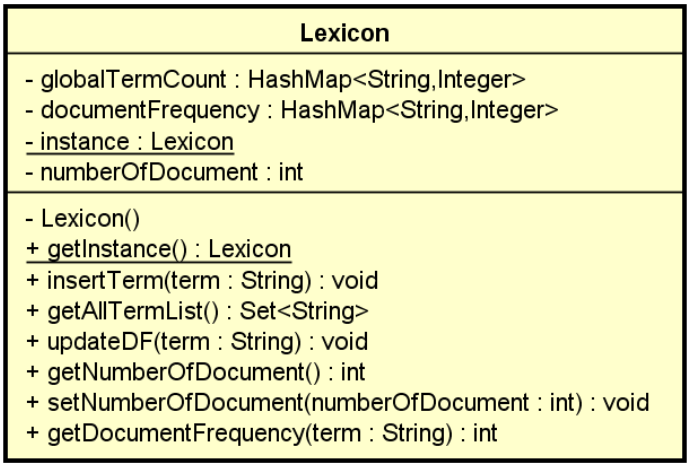


Gambar 4.8: Kelas *TFIDFWeighting*

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

- *calculateWeight*: *method* ini merupakan *method* yang diturunkan dari kelas *TermWeighting*. *Method* ini mengembalikan bobot dari *term* yang dihitung menggunakan teknik TF-IDF.
- *calculateTF*: *method* ini berfungsi untuk menghitung *TF* dari suatu *term*.
- *calculateIDF*: *method* ini berfungsi untuk menghitung *IDF* dari suatu *term*.

4.2.8 *Lexicon*



Gambar 4.9: Kelas *Lexicon*

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:

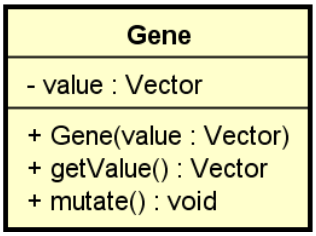
- *globalTermCount*: atribut ini bertipe *HashMap* dengan *key* bertipe *String* dan *value* bertipe *Integer*. Atribut ini berfungsi untuk menyimpan seluruh *term* yang muncul dan banyak kemunculannya dalam keseluruhan koleksi dokumen.
- *documentFrequency*: atribut ini bertipe *HashMap* dengan *key* bertipe *String* dan *value* bertipe *Integer*. Atribut ini berfungsi untuk menyimpan seluruh frekuensi dokumen dari tiap term.

- *instance*: atribut ini merupakan objek bertipe *Lexicon* sebagai instansiasi satu-satunya dari kelas *Lexicon* karena kelas ini bersifat *singleton*.
- *numberOfDocument*: atribut ini menyimpan banyaknya dokumen yang terdaftar di *Lexicon*.

Method yang ada pada kelas ini adalah:

- *Lexicon*: *method* ini merupakan *constructor private* untuk menjamin tidak akan ada lebih dari satu *instance* selama perangkat lunak berjalan.
- *getInstance*: *method* ini merupakan *method static* yang berfungsi sebagai *getter* dari atribut *instance*.
- *insertTerm*: *method* ini berfungsi untuk memasukkan *term* ke dalam atribut *globalTermCount*.
- *getAllTermList*: *method* ini bertugas mengembalikan daftar seluruh *term* yang pernah muncul di seluruh koleksi dokumen.
- *updateDF*: *method* ini berfungsi untuk menambah nilai TF dari *term* "*term*".
- *getNumberOfDocument*: *method* ini merupakan *getter* dari atribut *numberOfDocument*.
- *setNumberOfDocument*: *method* ini merupakan *setter* dari atribut *numberOfDocument*.
- *getDocumentFrequency*: *method* ini berfungsi untuk mendapatkan nilai DF dari *term* "*term*".

4.2.9 *Gene*

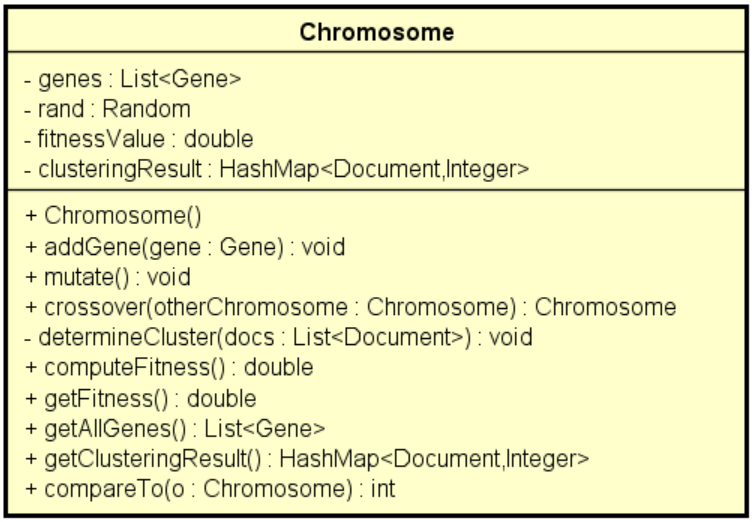


Gambar 4.10: Kelas *Gene*

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

- *Gene*: *method* ini merupakan *constructor* dari kelas *Gene* yang membutuhkan sebuah parameter bertipe *Vector* untuk mengisi atribut *value*.
- *getValue*: *method* ini merupakan *getter* dari atribut *value*.
- *mutate*: *method* ini berfungsi untuk melakukan mutasi pada gen. *Method* ini sebenarnya hanya bertugas memanggil fungsi *mutate()* dari atribut *value*.

4.2.10 *Chromosome*



Gambar 4.11: Kelas *Chromosome*

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

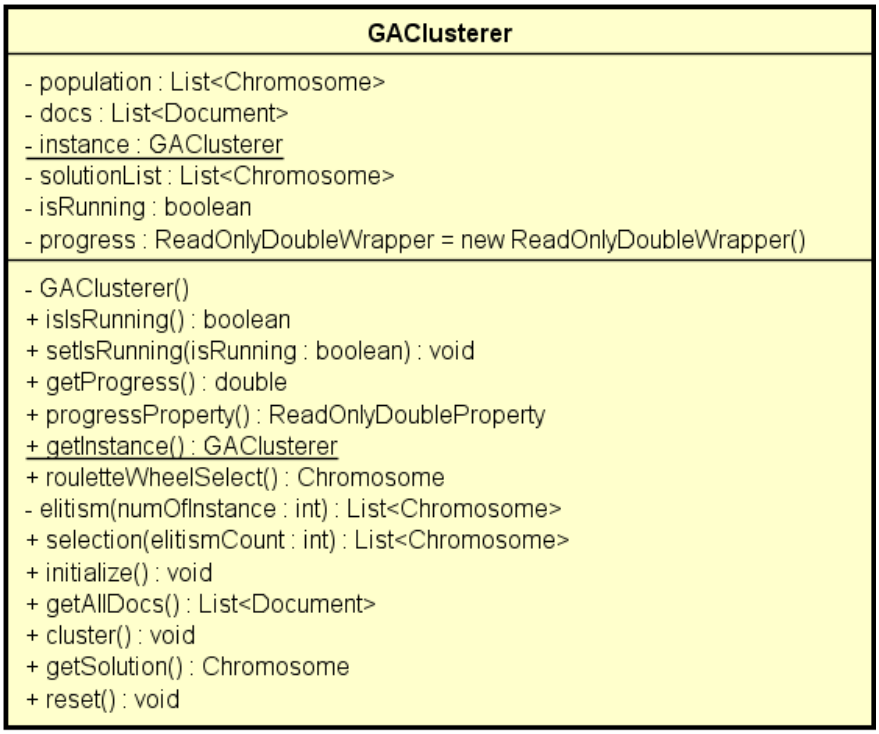
- *genes*: atribut bertipe *List of Gene* dan merupakan kumpulan gen yang terdapat dalam kromosom.
- *rand*: atribut ini merupakan objek *Random* milik *Java* dan berfungsi untuk membangkitkan bilangan acak yang dibutuhkan dalam setiap proses dalam kromosom.
- *fitnessValue*: atribut ini menyimpan nilai *fitness* dari kromosom.
- *clusteringResult*: atribut ini akan menyimpan hasil dari pengelompokan. Atribut ini bertipe *HashMap* yang menyimpan pasangan dokumen dan *cluster* dari dokumen tersebut.

Method yang terdapat dalam kelas ini adalah:

- *Chromosome*: *method* ini merupakan *constructor* tanpa parameter untuk membentuk objek dari kelas *Chromosome*.
- *addGene*: *method* ini bertugas untuk menambahkan satu gen ke dalam kromosom (ke dalam atribut *genes*).
- *mutate*: *method* ini berfungsi untuk melakukan mutasi pada kromosom dengan cara melakukan mutasi pada sebuah gen secara acak (Subbab 2.3.4).
- *crossover*: *method* ini bertugas untuk melakukan persilangan dengan kromosom *otherChromosome* untuk mengahsilkan keturunan (Subbab 2.3.3).
- *determineCluster*: *method* ini berfungsi untuk menentukan keanggotaan dari setiap dokumen.

- *computeFitness*: *method* ini mengembalikan nilai *fitness* dari kromosom (Subbab **??**).
- *getFitness*: *method* ini merupakan *getter* dari atribut *fitnessValue*.
- *getAllGenes*: *method* ini merupakan *getter* dari atribut *genes*.
- *getClusteringResult*: *method* ini merupakan *getter* dari atribut *clusteringResult*.
- *compareTo*: *method* ini merupakan turunan dari *interface Comparable* milik *Java* yang dibutuhkan untuk membandingkan kromosom ini dengan kromosom *o*. Method ini nantinya akan digunakan dalam proses *sorting*.

4.2.11 *GAClusterer*



Gambar 4.12: Kelas *GAClusterer*

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

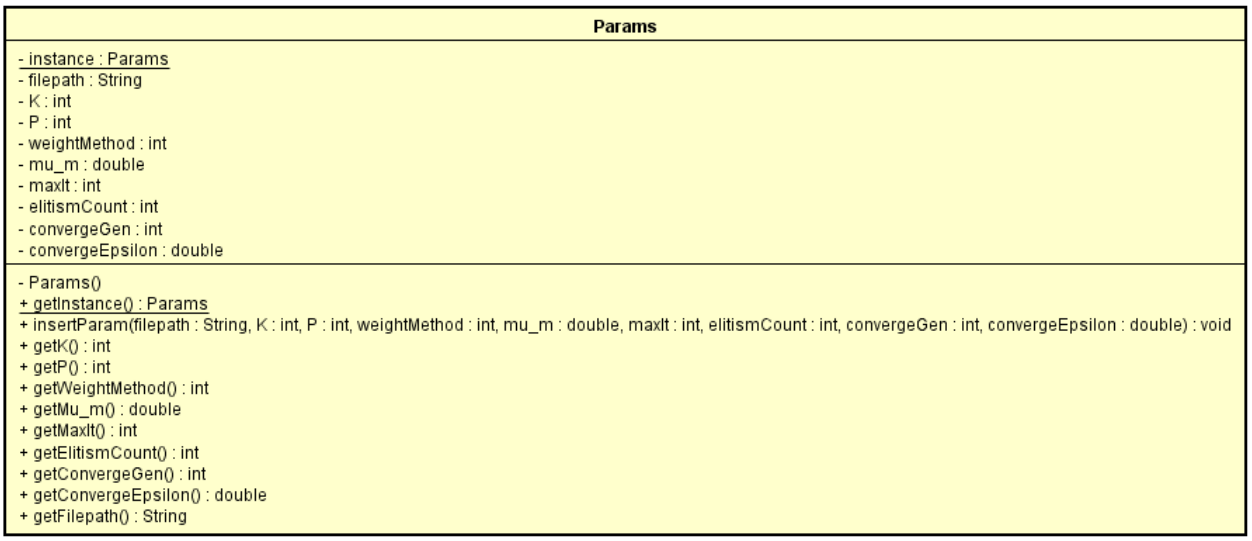
- *population*: atribut ini bertipe *List of Chromosome* yang merepresentasikan populasi pada generasi saat ini.
- *docs*: atribut ini bertipe *List of Document* yang berfungsi untuk menyimpan seluruh koleksi dokumen.
- *instance*: atribut *static* ini berfungsi untuk menyimpan *instance* dari kelas *GAClusterer*.

- *solutionList*: atribut ini bertipe *List of Chromosome* yang mencatat kromosom dengan nilai *fitness* terbaik untuk setiap generasinya.
- *isRunning*: atribut ini berfungsi untuk menyimpan status dari operasi pengelompokan menggunakan GA. Apabila bernilai *true* maka program sedang berjalan dan *false* apabila tidak sedang berjalan.
- *progress*: atribut ini bertipe *ReadOnlyDoubleWrapper* dan berfungsi untuk menyimpan perkembangan dari pengerjaan tugas pengelompokan ini ke antarmuka pengguna.

Method yang terdapat dalam kelas ini adalah:

- *GAClusterer*: *method* ini merupakan *constructor private* yang berfungsi untuk menjamin tidak akan ada *instance* dibuat diluar dari kelas ini.
- *setIsRunning*: *method* ini mrupakan *setter* dari atribut *isRunning*.
- *getProgress*: *method* ini mengembalikan perkembangan pekerjaan program (skala 0 sampai dengan 1).
- *progressProperty*: *method* ini merupakan *getter* dari atribut *progress*.
- *getInstance*: *method* ini merupakan *getter* dari atribut *instance*.
- *rouletteWheelSelect*: *method* ini bertugas untuk memilih sebuah kromosom menggunakan teknik *roulette-wheel selection* dari populasi.
- *elitism*: *method* ini bertugas untuk memilih n kromosom elit yang akan langsung masuk ke generasi berikutnya.
- *selection*: *method* ini bertugas untuk melakukan *roulette-wheel selection* sebanyak populasi untuk menghasilkan populasi dari generasi selanjutnya.
- *initialize*: *method* ini berfungsi untuk melakukan *indexing* dokumen dan membentuk populasi awal.
- *getAllDocs*: *method* ini merupakan *getter* dari atribut *docs*.
- *cluster*: *method* ini merupakan *method* utama yang bertugas melakukan pengelompokan dokumen dengan menggunakan algoritma genetika.
- *getSolution*: *method* ini berfungsi untuk menggembalikan solusi dari proses pengelompokan menggunakan algoritma genetika.
- *reset*: *method* ini berfungsi untuk mengatur ulang seluruh atribut untuk proses pengelompokan berikutnya.

4.2.12 *Params*



Gambar 4.13: Kelas *Params*

Kelas ini berfungsi untuk menyimpan seluruh parameter yang diberikan oleh pengguna agar dapat digunakan oleh setiap kelas yang membutuhkannya. Kelas ini bersifat *singleton* sehingga hanya akan ada satu buah *instance* selama perangkat lunak berjalan. Atribut yang ada dalam kelas ini adalah:

- *instance*: atribut ini merupakan objek bertipe *Params* sebagai instansiasi satu-satunya dari kelas *Params* karena kelas ini bersifat *singleton*.
- *filepath*: atribut ini berfungsi untuk menyimpan alamat dari direktori dokumen yang akan dikelompokkan.
- K: atribut ini berfungsi untuk menyimpan banyaknya *cluster* yang akan dipentuk dalam proses pengelompokan.
- P: atribut ini berfungsi untuk menyimpan banyaknya populasi yang akan dibentuk dalam proses pengelompokan menggunakan algoritma genetika.
- *weightingMethod*: atribut ini berfungsi untuk menyimpan metode pembobotan yang akan digunakan dalam proses pengelompokan. Apabila atribut ini bernilai 0 maka metode yang digunakan adalah TF-IDF sedangkan apabila bernilai 1 maka metode yang akan digunakan adalah bobot frekuensi.
- *mu_m*: atribut ini berfungsi untuk menyimpan probabilitas mutasi dalam bentuk bilangan riil bernilai antara 0 sampai dengan 1.
- *maxIt*: atribut ini berfungsi untuk menyimpan banyaknya iterasi maksimal yang dapat dilakukan.

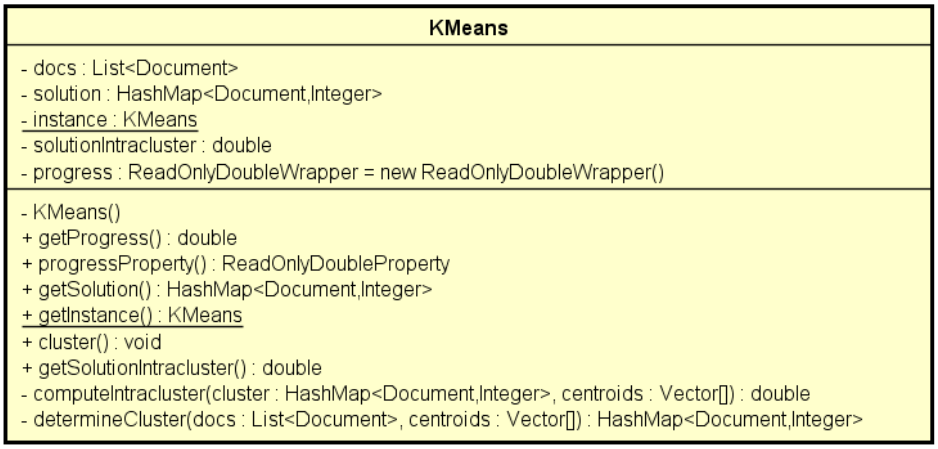
- *elitismCount*: atribut ini berfungsi untuk menyimpan banyaknya individu yang akan dijadikan elit pada tahap seleksi.
- *convergeGen*: atribut ini berfungsi untuk menyimpan banyaknya generasi konvergen sebelum proses pengelompokan diberhentikan.
- *convergeEpsilon*: atribut ini berfungsi untuk menyimpan nilai yang akan digunakan untuk membandingkan *fitness* tiap solusi dari setiap generasi untuk menentukan apakah sudah tercapai konvergen atau belum.

Method yang ada pada kelas ini adalah:

- *Params*: *method* ini merupakan *constructor private* untuk menjamin tidak akan ada lebih dari satu *instance* selama perangkat lunak berjalan.
- *getInstance*: *method* ini merupakan *method static* yang berfungsi sebagai *getter* dari atribut *instance*.
- *insertParam*: *method* ini bertugas untuk memasukkan atau mengubah nilai dari setiap atribut dalam kelas ini.
- *getK*: *method* ini merupakan *getter* dari atribut *K*.
- *getP*: *method* ini merupakan *getter* dari atribut *P*.
- *getWeightingMethod*: *method* ini merupakan *getter* dari atribut *weightingMethod*.
- *getMu_m*: *method* ini merupakan *getter* dari atribut *mu_m*.
- *getMaxIt*: *method* ini merupakan *getter* dari atribut *maxIt*.
- *getElitismCount*: *method* ini merupakan *getter* dari atribut *elitismCount*.
- *getConvergeGen*: *method* ini merupakan *getter* dari atribut *convergeGen*.
- *getConvergeEpsilon*: *method* ini merupakan *getter* dari atribut *convergeEpsilon*.
- *getFilepath*: *method* ini merupakan *getter* dari atribut *filepath*.

.

4.2.13 *KMeans*



Gambar 4.14: Kelas *KMeans*

Kelas ini merupakan kelas utama yang akan mengatur jalannya proses pengelompokan menggunakan algoritma *K-means*. Kelas ini merupakan kelas *singleton*. Atribut yang terdapat dalam kelas ini adalah:

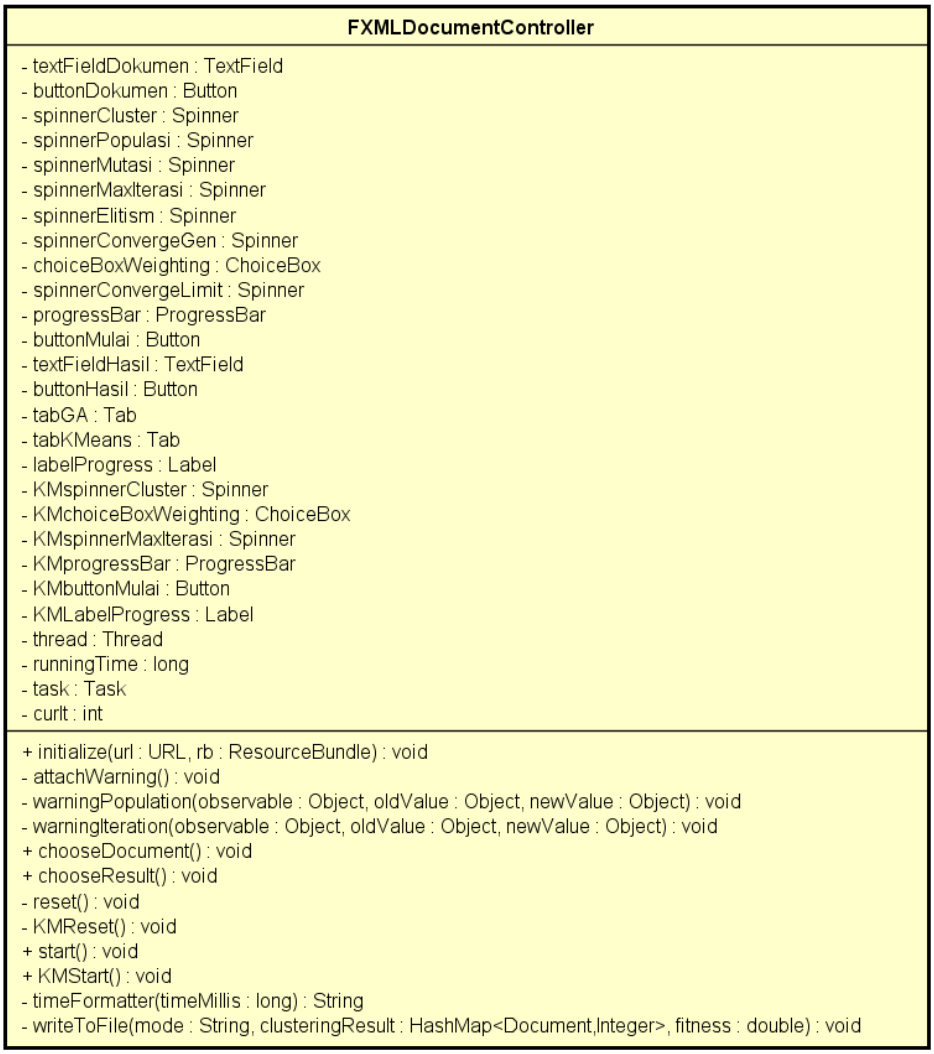
- *docs*: atribut ini bertipe *List of Document* yang berfungsi untuk menyimpan seluruh koleksi dokumen.
- *solution*: atribut ini berfungsi untuk menyimpan hasil pengelompokan dalam bentuk himpunan pasangan dokumen dan keanggotaan *cluster* dari dokumen tersebut.
- *instance*: atribut *static* ini berfungsi untuk menyimpan *instance* dari kelas *KMeans*.
- *solutionIntracluster*: atribut ini berfungsi untuk menyimpan nilai *intracluster* dari solusi yang telah didapatkan dari proses pengelompokan.
- *isRunning*: atribut ini berfungsi untuk menyimpan status dari operasi pengelompokan menggunakan GA. Apabila bernilai *true* maka program sedang berjalan dan *false* apabila tidak sedang berjalan.
- *progress*: atribut ini bertipe *ReadOnlyDoubleWrapper* dan berfungsi untuk menyimpan perkembangan dari pengerjaan tugas pengelompokan ini ke antarmuka pengguna.

Method yang terdapat dalam kelas ini adalah:

- *KMeans*: *method* ini merupakan *constructor private* yang berfungsi untuk menjamin tidak akan ada *instance* dibuat diluar dari kelas ini.
- *setIsRunning*: *method* ini mrupakan *setter* dari atribut *isRunning*.
- *getProgress*: *method* ini mengembalikan perkembangan pekerjaan program (skala 0 sampai dengan 1).

- *progressProperty*: *method* ini merupakan *getter* dari atribut *progress*.
- *getSolution*: *method* ini merupakan *getter* dari atribut *solution*.
- *getInstance*: *method* ini merupakan *getter* dari atribut *instance*.
- *cluster*: *method* ini merupakan *method* utama yang bertugas melakukan pengelompokan dokumen dengan menggunakan algoritma *K-means*.
- *getSolutionIntracluster*: *method* ini merupakan *getter* dari atribut *solutionIntracluster*.
- *computeIntracluster*: *method* ini bertugas untuk menghitung *intracluster* apabila diketahui *centroid* setiap *cluster* adalah *centroids* dan keanggotaan setiap dokumen adalah *cluster*.
- *determineCluster*: *method* ini berfungsi untuk menentukan keanggotaan dari setiap dokumen.

4.2.14 *FXMLDocumentController*



Gambar 4.15: Kelas *FXMLDocumentController*

Kelas ini merupakan kelas yang bertugas untuk mengendalikan seluruh aktivitas yang ada di antarmuka dan menghubungkannya dengan kelas lain yang dibutuhkan. Atribut yang terdapat dalam kelas ini adalah:

- *textFieldDokumen*: atribut ini bertipe *TextField* dan berfungsi menampilkan direktori dokumen pada antarmuka pengguna.
- *buttonDokumen*: atribut ini bertipe *Button* yang merupakan tombol untuk memilih direktori dokumen pada antarmuka pengguna.
- *spinnerCluster*: atribut ini bertipe *Spinner* yang akan menangani masukan untuk parameter banyaknya *cluster* pada antarmuka pengguna di bagian GA.
- *spinnerPopulasi*: atribut ini bertipe *Spinner* yang akan menangani masukan untuk parameter banyaknya populasi pada antarmuka pengguna di bagian GA.
- *spinnerMutasi*: atribut ini bertipe *Spinner* yang akan menangani masukan untuk parameter probabilitas mutasi pada antarmuka pengguna di bagian GA.
- *spinnerMaxIterasi*: atribut ini bertipe *Spinner* yang akan menangani masukan untuk parameter maksimum iterasi pada antarmuka pengguna di bagian GA.
- *spinnerElitism*: atribut ini bertipe *Spinner* yang akan menangani masukan untuk parameter individu elitisme pada antarmuka pengguna di bagian GA.
- *spinnerConvergeGen*: atribut ini bertipe *Spinner* yang akan menangani masukan untuk parameter banyaknya generasi konvergen pada antarmuka pengguna di bagian GA.
- *choiceBoxWeighting*: atribut ini bertipe *ChoiceBox* yang akan menangani masukan untuk parameter metode pembobotan pada antarmuka pengguna di bagian GA.
- *spinnerConvergeLimit*: atribut ini bertipe *Spinner* yang akan menangani masukan untuk parameter banyaknya generasi konvergen pada antarmuka pengguna di bagian GA.
- *progressBar*: atribut ini bertipe *progressBar* yang akan menampilkan perkembangan dari proses pengelompokan menggunakan algoritma GA.
- *buttonMulai*: atribut ini bertipe *Button* yang merupakan tombol untuk memulai proses pengelompokan menggunakan algoritma GA.
- *textFieldHasil*: atribut ini bertipe *TextField* dan berfungsi menampilkan direktori hasil pada antarmuka pengguna.
- *buttonHasil*: atribut ini bertipe *Button* yang merupakan tombol untuk memilih direktori hasil pada antarmuka pengguna.
- *tabGA*: atribut ini bertipe *Tab* yang merupakan *tab* untuk memilih pengelompokan menggunakan algoritma GA.

- *tabKMeans*: atribut ini bertipe *Tab* yang merupakan *tab* untuk memilih pengelompokan menggunakan algoritma *K-means*.
- *labelProgress*: atribut ini bertipe *Label* yang berfungsi untuk menampilkan status dari proses pengelompokan menggunakan algoritma GA.
- *KMspinnerCluster*: atribut ini bertipe *Spinner* yang akan menangani masukan untuk parameter banyaknya *cluster* pada antarmuka pengguna di bagian *K-means*.
- *KMchoiceBoxWeighting*: atribut ini bertipe *ChoiceBox* yang akan menangani masukan untuk parameter metode pembobotan pada antarmuka pengguna di bagian *K-means*.
- *KMspinnerMaxIterasi*: atribut ini bertipe *Spinner* yang akan menangani masukan untuk parameter maksimum iterasi pada antarmuka pengguna di bagian *K-means*.
- *KMprogressBar*: atribut ini bertipe *progressBar* yang akan menampilkan perkembangan dari proses pengelompokan menggunakan algoritma *K-means*.
- *KMbuttonMulai*: atribut ini bertipe *Button* yang merupakan tombol untuk memulai proses pengelompokan menggunakan algoritma *K-means*.
- *labelProgress*: atribut ini bertipe *Label* yang berfungsi untuk menampilkan status dari proses pengelompokan menggunakan algoritma *K-means*.
- *thread*: atribut ini bertipe *Thread* dan merupakan sebuah *thread* yang akan menjalankan proses pengelompokan.
- *runningTime*: atribut ini bertipe *long* dan berfungsi untuk menyimpan lamanya program berjalan dalam milidetik.
- *task*: atribut ini bertipe *Task* dan berfungsi untuk menjalankan tugas pengelompokan.
- *curIt*: atirbut ini bertipe *int* dan berfungsi untuk menyimpan banyaknya iterasi saat ini.

Method yang terdapat dalam kelas ini adalah:

- *initialize*: *method* ini berfungsi untuk menginisialisasi seluruh atribut dan nilai awalnya pada antarmuka pengguna.
- *attachWarning*: *method* ini berfungsi untuk menambahkan *listener* sehingga dapat menampilkan pesan apabila aturan validasi dari suatu masukan dilanggar.
- *warningPopulation*: *method* ini berfungsi untuk melakukan pengecekkan terhadap aturan validasi parameter banyaknya populasi dan individu elitisme lalu menampilkan pesan apabila aturan validasi dilanggar.
- *warningIteration*: *method* ini berfungsi untuk melakukan pengecekkan terhadap aturan validasi parameter maksimal iterasi dan banyaknya generasi konvergen lalu menampilkan pesan apabila aturan validasi dilanggar.

- *chooseDocument*: *method* ini berfungsi untuk memilih direktori dokumen.
- *chooseResult*: *method* ini berfungsi untuk memilih direktori hasil.
- *reset*: *method* ini berfungsi untuk mengembalikan kondisi dari seluruh objek dalam halaman GA agar bisa kembali digunakan untuk proses pengelompokan selanjutnya.
- *KMReset*: *method* ini berfungsi untuk mengembalikan kondisi dari seluruh objek dalam halaman *K-means* agar bisa kembali digunakan untuk proses pengelompokan selanjutnya.
- *start*: *method* ini berfungsi untuk memulai proses pengelompokan menggunakan GA.
- *KMStart*: *method* ini berfungsi untuk memulai proses pengelompokan menggunakan *K-means*.
- *timeFormatter*: *method* ini berfungsi untuk membentuk *string* "*hh* jam *mm* menit *ss* detik" berdasarkan input waktu dalam milidetik.
- *writeToFile*: *method* ini berfungsi untuk menulis hasil pengelompokan ke dalam suatu *file* dengan ekstensi CSV.

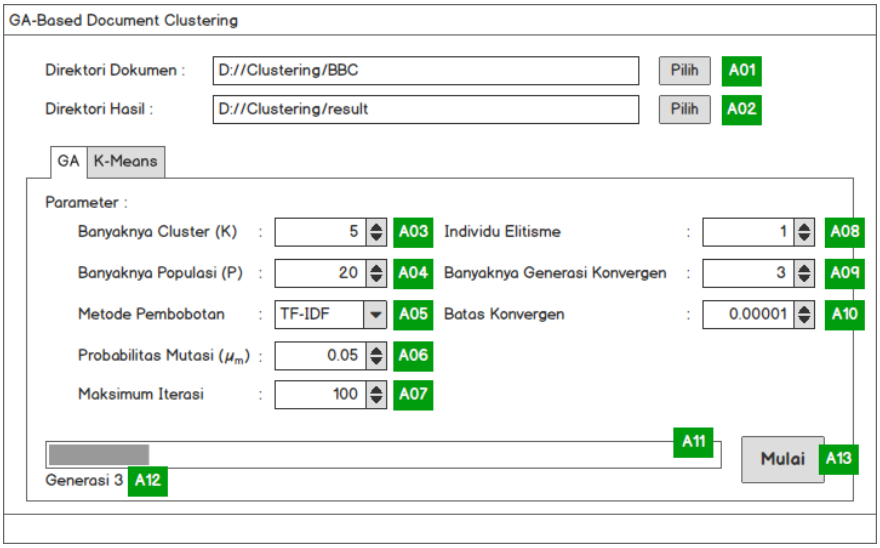
4.3 Perancangan Antarmuka Pengguna

Antarmuka yang dirancang untuk perangkat lunak ini hanya terdiri dari 2 halaman. Rancangan antarmuka ini dibuat sedemikian rupa sehingga memudahkan penggunannya dalam melakukan pengujian terhadap perangkat lunak. 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.

4.3.1 Halaman Algoritma Genetika

Gambar 4.16 menunjukkan halaman yang dapat digunakan oleh pengguna untuk mengelompokan dokumen menggunakan algoritma genetika. Pada halaman ini terdapat beberapa *field* yang dapat digunakan pengguna untuk mengatur nilai dari masing-masing parameter.

[1] https://balsamiq.com/



Gambar 4.16: Rancangan antarmuka halaman algoritma genetika

Penjelasan setiap *field* dalam halaman ini dijelaskan dalam Tabel 4.2.

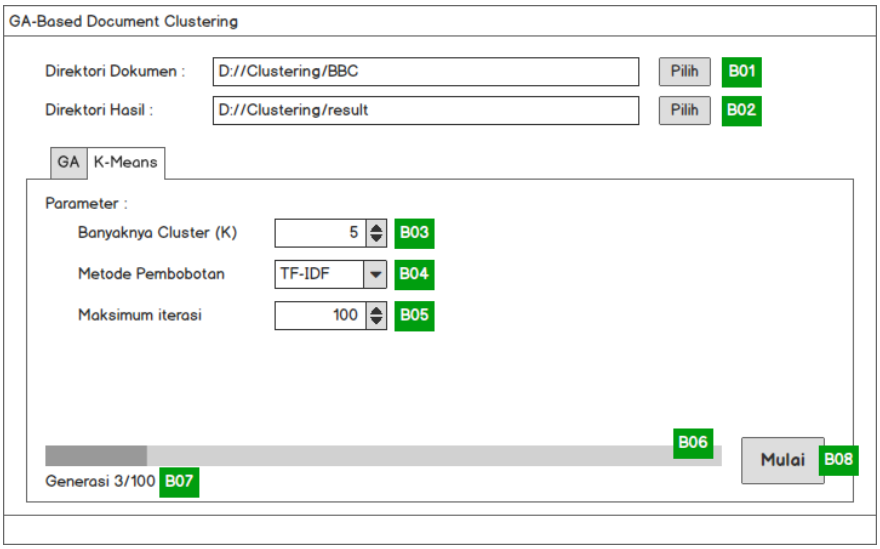
Kode	**Nama**	**Jenis**	***Defaut value***	**Wajib**	**Aturan validasi**
A01	Direktori Dokumen	*file chooser*	-	ya	-
A02	Direktori Hasil	*file chooser*	-	ya	-
A03	Banyaknya Cluster	*spinner*	5	ya	Nilai minimum 1
A04	Banyaknya Populasi	*spinner*	1	ya	Nilai minimum 1 dan harus lebih besar dari Individu Elitisme (A08)
A05	Metode Pembobotan	*dropdown*	TF-IDF	ya	-
A06	Probabilitas Mutasi	*spinner*	0.05	ya	Nilai minimum 0.01, maksimum 1.00
A07	Maksimum Iterasi	*spinner*	100	ya	Nilai minimum 1 dan harus lebih besar dari Banyaknya Generasi Konvergen (A09)
A08	Individu Elitisme	*spinner*	1	ya	Nilai minimum 0 dan harus lebih kecil dari Banyaknya Populasi (A04)
A09	Banyaknya Generasi Konvergen	*spinner*	3	ya	Nilai minimum 2 dan harus lebih kecil dari Maksimum Iterasi (A07)
A10	Batas Konvergen	*spinner*	0.00001	ya	Hanya bisa bernilai 10^{-3}, 10^{-4}, 10^{-5}, 10^{-6}, dan 10^{-7}

Tabel 4.2: Rincian *field* pada halaman algoritma genetika

Objek dengan kode A11 merupakan sebuah *progress bar* yang akan menampilkan perkembangan dari jalannya program untuk setiap iterasi. Apabila proses dalam suatu iterasi sudah selesai, maka akan mengubah nilai dari label A12 dan membuat *progress bar* kembali kosong. Label dengan kode A12 berfungsi untuk menampilkan status dari program yang sedang berjalan. Label ini tidak akan memiliki nilai apabila program belum dijalankan dan akan menampilkan status "Inisialisasi..." apabila program sedang melakukan pengindeksan dokumen dan inisialisasi populasi. Setelah iterasi dimulai maka label A12 akan menampilkan generasi saat ini. Sebagai contoh apabila label A12 menampilkan status "Generasi 1" artinya saat ini program sedang melakukan proses-proses pada

generasi 1. Begitu juga untuk "Generasi 2" dan seterusnya. Tombol dengan kode A13 berfungsi untuk memulai proses pengelompokan menggunakan algoritma genetika berdasarkan parameter yang telah dimasukkan.

4.3.2 Halaman *K-Means*



Gambar 4.17: Rancangan antarmuka halaman algoritma genetika

Gambar 4.17 merupakan tampilan yang akan digunakan oleh pengguna untuk melakukan pengelompokan dokumen menggunakan algoritma *K-means*. Berbeda dengan halaman algoritma genetika, pada halaman ini hanya terdapat tiga buah parameter yang dapat diubah-ubah oleh pengguna dalam melakukan pengelompokan.

Kode	**Nama**	**Jenis**	***Defaut value***	**Wajib**	**Aturan validasi**
B01	Direktori Dokumen	*file chooser*	-	ya	-
B02	Direktori Hasil	*file chooser*	-	ya	-
B03	Banyaknya Cluster	*spinner*	5	ya	Nilai minimum 1
B04	Metode Pembobotan	*dropdown*	TF-IDF	ya	-
B05	Maksimum Iterasi	*spinner*	100	ya	Nilai minimum 1

Tabel 4.3: Rincian *field* pada halaman algoritma *K-means*

Dalam Tabel 4.3, objek dengan kode B06 merupakan sebuah *progress bar* yang akan menampilkan perkembangan dari jalannya program yang menyatakan banyaknya iterasi yang sudah selesai dijalankan dibandingkan dengan maksimum iterasi. Label B07 akan berisi keterangan dari proses

yang sedang berlangsung. Salah satu contoh isi dari label B07 adalah "Iterasi 2/100" yang berarti saat ini sedang dilakukan pemrosesan pada iterasi kedua dari 100 iterasi. Tombol dengan kode B08 berfungsi untuk memulai proses pengelompokan menggunakan algoritma *K-means* berdasarkan parameter yang telah dimasukkan.

BAB 5

PENGUJIAN DAN EKSPERIMEN

5.1 Skenario Pengujian Eksperimental

Eksperimen dilakukan dengan menggunakan spesifikasi komputer sebagai berikut:

1. Tipe *processor*: Intel(R) Core(TM) i7-4720HQ CPU @2.60GHz
2. Memori: 12288MB RAM
3. Sistem operasi: Windows 10 Pro 64-bit (10.0, Build 17763)

Pada suatu penelitian, ada beberapa jenis variabel yang akan diteliti diantaranya variabel bebas, variabel terikat, dan variabel kontrol. Pada penelitian ini, pengujian eksperimental dilakukan dengan tujuan untuk mengetahui hubungan antara parameter masukan dengan waktu dan hasil pengelompokan. Oleh karena itu, variabel bebas dari penelitian ini merupakan variabel yang menjadi parameter masukan yang terdapat pada antarmuka pengguna (Subbab 4.3). Namun, tidak semua parameter masukan yang terdapat pada antarmuka pengguna akan menjadi variabel bebas. Parameter masukan yang merupakan variabel bebas pada penelitian ini diantaranya adalah:

1. Banyaknya populasi
2. Metode pembobotan
3. Probabilitas mutasi
4. Individu elitisme

Selain variabel bebas, terdapat juga variabel kontrol yang merupakan parameter masukan pada antarmuka pengguna. Variabel kontrol dalam penelitian ini adalah sebagai berikut:

- Banyaknya *cluster*
- Maksimum iterasi
- Banyaknya generasi konvergen
- Batas konvergen

Keempat variabel ini tidak masuk ke dalam variabel bebas karena alasan tertentu. Variabel banyaknya *cluster* akan dijadikan variabel kontrol karena nilai dari variabel ini harus disesuaikan dengan banyaknya topik pada *dataset* (Subbab 3.1) untuk mendapatkan hasil yang maksimal. Variabel maksimum iterasi juga merupakan variabel kontrol untuk menjaga agar proses pengelompokan tidak menjadi sangat lama karena terlalu banyak iterasi yang terjadi. Sama halnya untuk variabel banyaknya generasi konvergen dan batas konvergen, kedua variabel ini juga dapat menjaga agar iterasi yang terjadi tidak terlalu banyak sehingga menghabiskan waktu yang lebih banyak. Variabel terikat dalam penelitian ini adalah sebagai berikut:

- Waktu
- Nilai *intracluster*
- Banyak iterasi
- Nilai *purity*

Keempat variabel terikat ini merupakan variabel yang dipengaruhi oleh berubahnya variabel bebas. Dalam penelitian ini hanya ada empat aspek yang diperhatikan yaitu waktu, nilai *intracluster*, banyak iterasi, dan nilai *purity*. Oleh karena itu, akan dibuat suatu skenario pengujian eksperimental untuk mengetahui pengaruh variabel bebas terhadap variabel terikat. Variasi nilai dari variabel bebas ditunjukkan dalam Tabel 5.1.

Variabel bebas	Variasi		
Banyaknya Populasi	50	100	150
Metode Pembobotan	TF-IDF	Frekuensi	
Probabilitas Mutasi	0	0.05	0.25
Individu Elitisme	0	1	5

Tabel 5.1: Variasi nilai variabel bebas

Setiap variabel dalam Tabel 5.1 memiliki dua sampai tiga variasi nilai. Karena keterbatasan waktu, maka pengujian eksperimental ini tidak akan menguji seluruh kombinasi parameter yang ada. Variasi dari parameter dalam Tabel 5.1 hanya akan menggantikan nilai pada suatu kasus uji yang disebut kasus uji ideal. Kasus uji ideal merupakan kasus uji yang dianggap akan menghasilkan keluaran yang ideal. Kasus uji ideal untuk pengujian eksperimental dalam penelitian ini adalah sebagai berikut:

- Banyaknya *Cluster*: 5
- Banyaknya Populasi: 100
- Metode Pembobotan: TF-IDF
- Probabilitas Mutasi: 0.05
- Maksimum Iterasi it: 100
- Individu Elitisme: 1

- Banyaknya Generasi Konvergen: 3
- Batas Konvergen: 0.00001

5.2 Eksperimen Algoritma Genetika

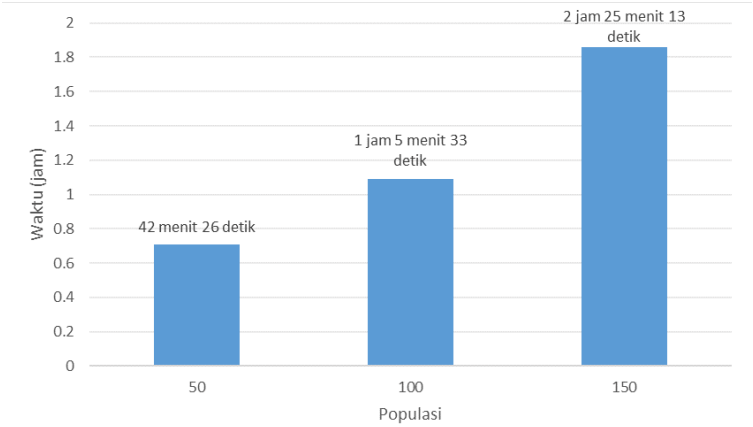
Eksperimen dibagi menjadi beberapa bagian berdasarkan parameter yang diubah. Terdapat 4 bagian yaitu berdasarkan keempat variabel bebas yang telah dijelaskan pada Subbab 5.1. Untuk setiap variasi parameter, dilakukan lima kali pengujian dan hasilnya akan dirata-rata. Berikut adalah hasil dari eksperimen untuk keempat variabel bebas.

1. Banyaknya populasi:

Populasi	Waktu (jam)	*Intracluster*	Iterasi	*Purity*
50	0.707222222	760.5179536	4.8	0.779595506
100	1.092666667	771.6192479	4.6	0.798382022
150	1.857611111	862.1796042	4.6	0.746696629

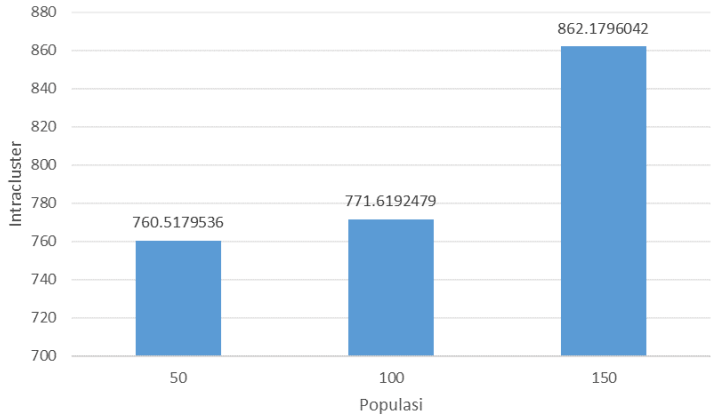
Tabel 5.2: Rata-rata hasil pengelompokan dengan variasi variabel banyaknya populasi

Berdasarkan Tabel 5.2, Kenaikan populasi berbanding lurus dengan kenaikan waktu pemrosesan dan *intracluster similarity*. Sedangkan untuk banyak iterasi dan nilai *purity* sama sekali tidak bergantung dengan kenaikan populasi. Dapat disimpulkan bahwa kenaikan populasi tidak mempengaruhi hasil pengelompokan, hanya meningkatkan waktu pemrosesan saja. Berdasarkan data pada Tabel 5.2, dibentuk empat buah grafik (Gambar 5.1 - 5.4) untuk menunjukan hubungan antara populasi dengan keempat variabel terikat.



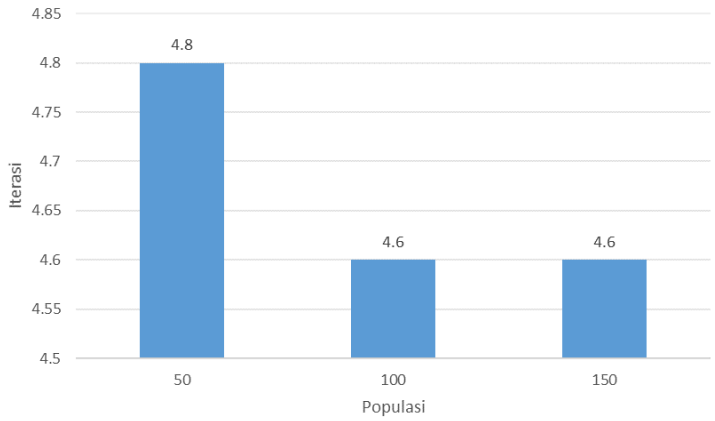
Gambar 5.1: Grafik hubungan banyaknya populasi dengan waktu pengelompokan

Dapat disimpulkan dari grafik pada Gambar 5.1, perbedaan waktu pada saat populasi berjumlah 100 dan 150 cukup signifikan dibandingkan dengan perbedaan waktu antara populasi berjumlah 50 dan 100. Hal ini disebabkan karena kompleksitas perangkat lunak ini tidaklah linear. Selain itu, semakin banyak populasi maka calon solusi akan semakin banyak pula. Hal ini menyebabkan GA akan lebih sulit mencapai konvergen.



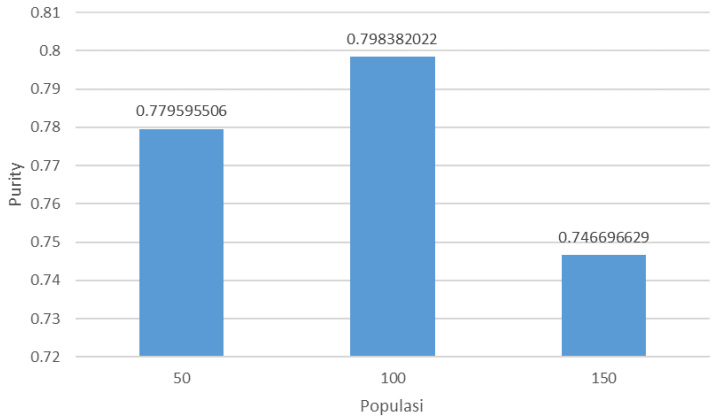
Gambar 5.2: Grafik hubungan banyaknya populasi dengan *intracluster similarity*

Dapat disimpulkan dari grafik pada Gambar 5.2, kenaikan populasi juga mempengaruhi kenaikan *intracluster similarity*. Sama dengan variabel waktu, perbedaan *intracluster similarity* antara populasi berjumlah 100 dan 150 jauh lebih besar dibandingkan perbedaan *intracluster similarity* antara populasi berjumlah 50 dan 100. Hal ini dikarenakan dengan populasi yang lebih banyak maka kemungkinan untuk mendapat solusi yang lebih baik akan semakin besar.



Gambar 5.3: Grafik hubungan banyaknya populasi dengan banyaknya iterasi

Dapat disimpulkan dari grafik pada Gambar 5.3, kenaikan populasi tidak mempengaruhi banyaknya iterasi yang dilakukan dalam mencapai konvergen. Rata-rata banyaknya iterasi untuk populasi berjumlah 50, 100, dan 150 tidak memiliki perbedaan yang begitu signifikan.



Gambar 5.4: Grafik hubungan banyaknya populasi dengan nilai *purity*

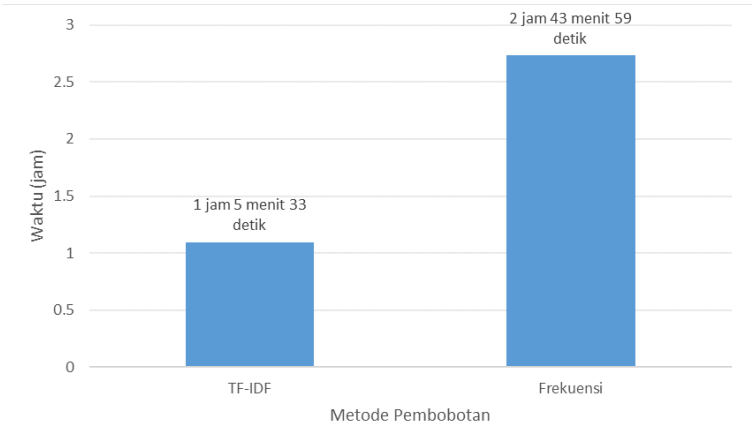
Dapat disimpulkan dari grafik pada Gambar 5.4, kenaikan populasi juga tidak mempengaruhi nilai *purity* sama seperti banyaknya iterasi. Berdasarkan nilai *purity*, maka hasil terbaik diperoleh dengan populasi sebanyak 100 individu.

2. Metode pembobotan:

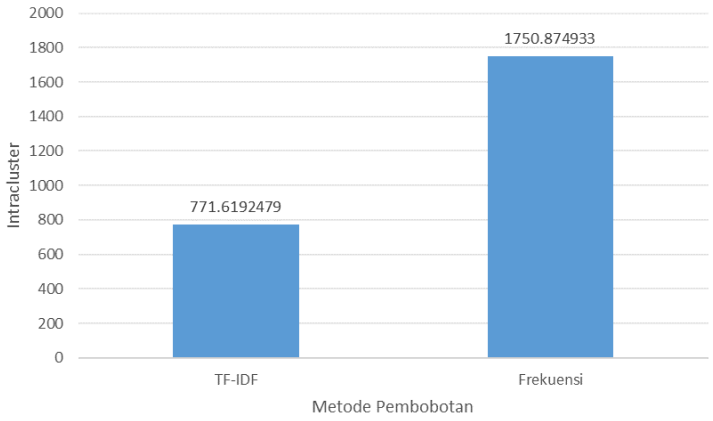
Bobot	Waktu (jam)	*Intracluster*	Iterasi	*Purity*
TF-IDF	1.092666667	771.6192479	4.6	0.798382022
Frekuensi	2.733222222	1750.874933	7	0.571325843

Tabel 5.3: Rata-rata hasil pengelompokan dengan variasi variabel metode pembobotan

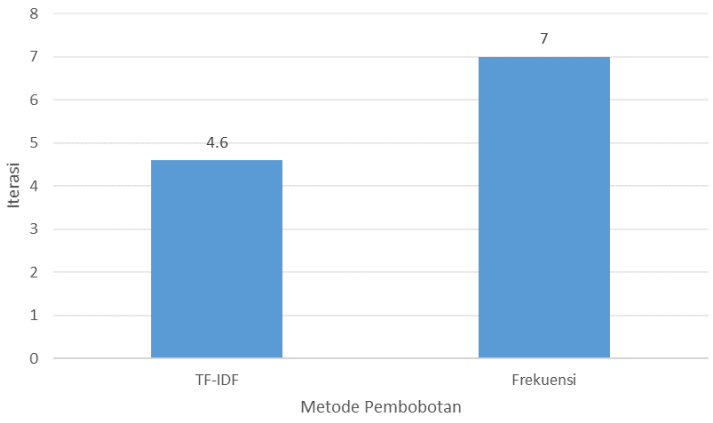
Berdasarkan Tabel 5.3, waktu yang diperlukan untuk pengelompokan menggunakan bobot TF-IDF jauh lebih cepat hingga 167% dibandingan dengan menggunakan bobot frekuensi. *Intracluster similarity* yang dihasilkan menggunakan bobot TF-IDF hanya 44% dari *intracluster similarity* yang dihasilkan apabila menggunakan bobot frekuensi. Hal ini terjadi karena bobot frekuensi dari suatu dokumen pasti lebih besar nilainya dibandingkan dengan bobot TF-IDF. Dengan menggunakan bobot frekuensi, banyaknya iterasi yang dilakukan 52% lebih banyak dibandingkan dengan menggunakan bobot TF-IDF. Nilai purity yang didapatkan apabila menggunakan bobot TF-IDF 40% lebih besar apabila dibandingkan dengan menggunakan bobot frekuensi. Berdasarkan Tabel 5.3, dibentuk empat buah grafik (Gambar 5.5 - 5.8) untuk menunjukan hubungan antara metode pembobotan dengan keempat variabel terikat.



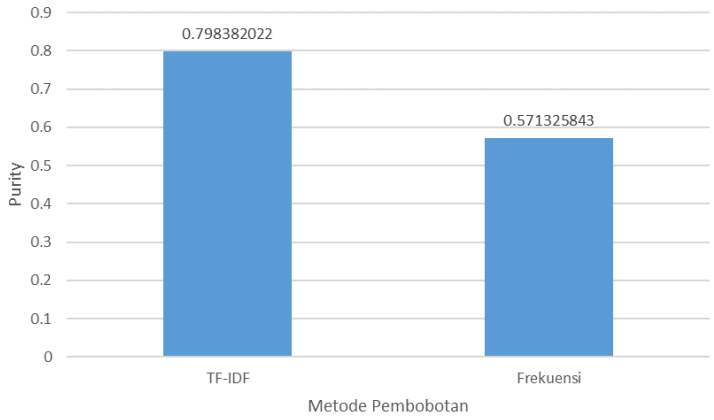
Gambar 5.5: Grafik hubungan metode pembobotan dengan waktu pengelompokan



Gambar 5.6: Grafik hubungan metode pembobotan dengan *intracluster similarity*



Gambar 5.7: Grafik hubungan metode pembobotan dengan banyaknya iterasi



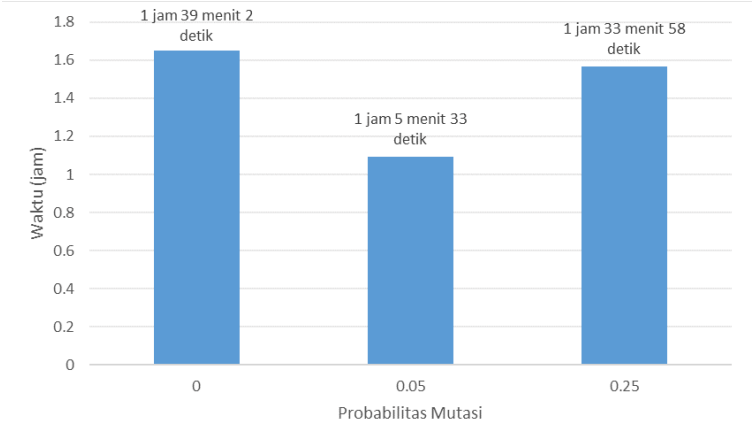
Gambar 5.8: Grafik hubungan metode pembobotan dengan nilai *purity*

3. Probabilitas mutasi:

Probabilitas mutasi	Waktu (jam)	*Intracluster*	Iterasi	*Purity*
0	1.650722222	1105.863837	5.2	0.632359551
0.05	1.092666667	771.6192479	4.6	0.798382022
0.25	1.566277778	1290.371505	5	0.437123596

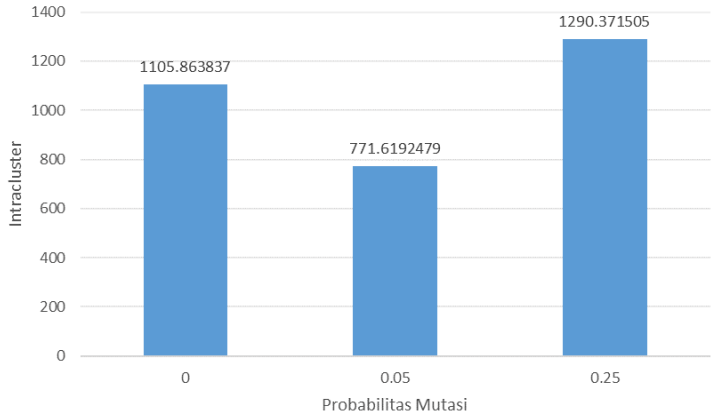
Tabel 5.4: Rata-rata hasil pengelompokan dengan variasi variabel probabilitas mutasi

Berdasarkan Tabel 5.4, dibentuk empat buah grafik (Gambar 5.9 - 5.12) untuk menunjukan hubungan antara metode pembobotan dengan keempat variabel terikat.



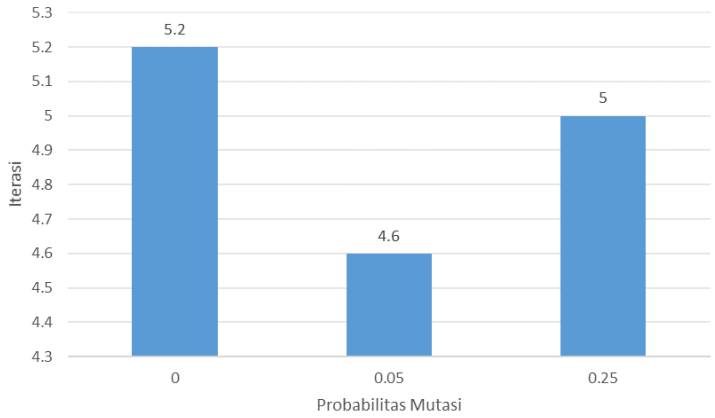
Gambar 5.9: Grafik hubungan probabilitas mutasi dengan waktu pengelompokan

Dapat disimpulkan dari grafik pada Gambar 5.9, waktu tempuh paling cepat didapatkan dengan menggunakan probabilitas mutasi 0.05. Apabila probabilitas mutasi bernilai 0, maka mutasi tidak pernah terjadi sehingga GA lebih sulit mencapai konvergen. Hal ini menyebabkan waktu yang dibutuhkan untuk mencapai konvergen lebih lama. Apabila probabilitas mutasi bernilai 0.25, mutasi terjadi terlalu sering sehingga proses pencarian menjadi kurang terarah seperti yang telah dijelaskan pada Subbab 2.3.4.



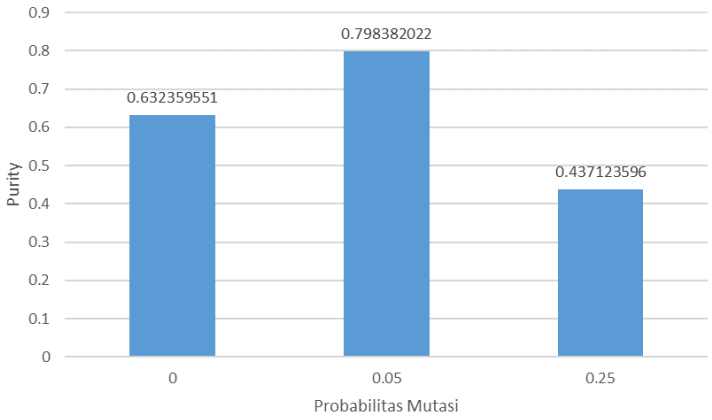
Gambar 5.10: Grafik hubungan probabilitas mutasi dengan *intracluster similarity*

Berdasarkan *intracluster similarity* pada grafik dalam Gambar 5.10, hasil pengelompokan dengan probabilitas mutasi 0 lebih baik 43% dari hasil menggunakan probabilitas mutasi 0.05. Hasil menggunakan probabilitas mutasi 0.25 juga lebih baik 67% dibandingkan dengan menggunakan probabilitas mutasi 0.05.



Gambar 5.11: Grafik hubungan probabilitas mutasi dengan banyaknya iterasi

Dapat disimpulkan dari grafik pada Gambar **??**, jumlah iterasi paling sedikit akan didapatkan dengan menggunakan probabilitas mutasi 0.05. Sama dengan waktu tempuh, apabila probabilitas mutasi bernilai 0, maka mutasi tidak pernah terjadi sehingga GA lebih sulit mencapai konvergen. Hal ini menyebabkan jumlah iterasi yang dibutuhkan untuk mencapai konvergen lebih banyak. Apabila mutasi terjadi terlalu sering (pada kasus ini dengan probabilitas 0.25), maka GA juga akan sulit mencapai konvergen karena proses pencariannya menjadi tidak terarah.



Gambar 5.12: Grafik hubungan probabilitas mutasi dengan nilai *purity*

Berdasarkan nilai *purity* pada grafik dalam Gambar 5.12, hasil terbaik didapatkan dengan menggunakan probabilitas mutasi 0.05. Nilai *purity* dengan menggunakan probabilitas mutasi 0.05 lebih baik 26% dibandingkan dengan menggunakan probabilitas mutasi 0 dan lebih baik 82% dibandingkan menggunakan probabilitas mutasi 0.25.

4. Individu elitisme:

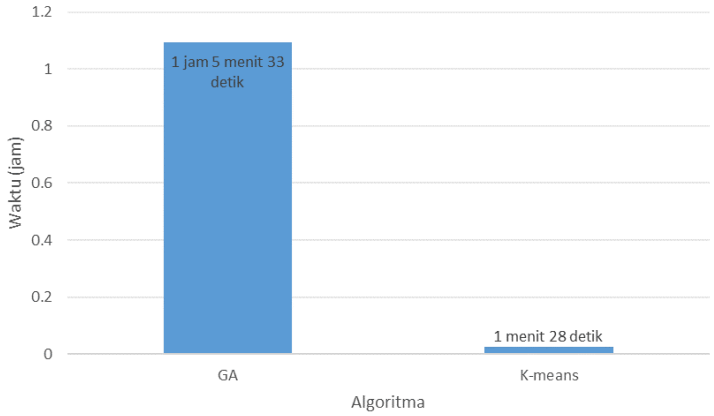
5.3 Eksperimen *K-Means*

Sebagai perbandingan terhadap pengelompokan menggunakan algoritma genetika, maka dibuat pula eksperimen menggunakan algoritma *K-means*. Seperti yang telah dijelaskan pada Subbab 4.3.2, pada penelitian ini *K-means* hanya memiliki tiga buah parameter. Hanya satu diantara ketiga parameter tersebut yang dapat dijadikan variabel bebas karena alasan yang sama dengan yang telah dijelaskan pada Subbab 5.1. Eksperimen menggunakan algoritma *K-means* hanya dilakukan menggunakan dua variasi yaitu dengan metode pembobotan TF-IDF dan Frekuensi. Namun dalam eksperimen pada penelitian ini, hanya variasi menggunakan bobot TF-IDF yang akan diuji. Hasil dari eksperimen menggunakan algoritma *K-means* ini kemudian akan dibandingkan dengan kasus uji ideal pada algoritma genetika. Berbeda dari pengujian menggunakan algoritma genetika, pengujian menggunakan *K-means* akan dilakukan sebanyak 10 kali tiap variasi. Hasil eksperimen menggunakan algoritma *K-means* dijelaskan dalam Tabel 5.5.

Algoritma	Waktu (jam)	*Intracluster*	Iterasi	*Purity*
GA	1.092666667	771.6192479	4.6	0.798382022
K-means	0.024472222	831.5881861	11.5	0.51191103

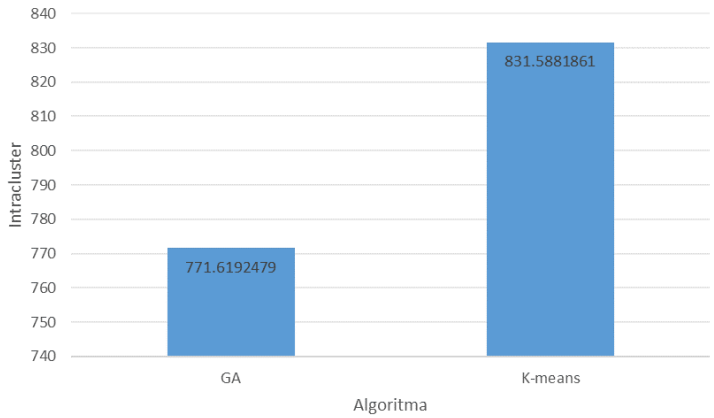
Tabel 5.5: Rata-rata hasil pengelompokan dengan menggunakan algoritma *K-means*

Berdasarkan Tabel 5.5, dibentuk empat grafik pada Gambar 5.13 - 5.16 untuk menjelaskan perbandingan antara algoritma genetika dengan algoritma *K-means* berdasarkan keempat variabel terikat.



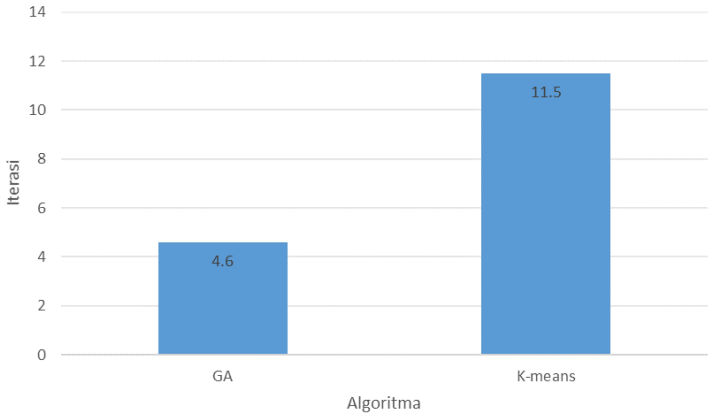
Gambar 5.13: Grafik hubungan algoritma dengan waktu tempuh

Berdasarkan grafik pada Gambar 5.13, waktu yang dibutuhkan algoritma genetika lebih banyak sekitar 4365% dibandingkan dengan algoritma *K-means*. Hal ini dikarenakan proses komputasi yang dilakukan pada algoritma genetika jauh lebih banyak dan kompleks dibandingkan dengan algoritma *K-means*.



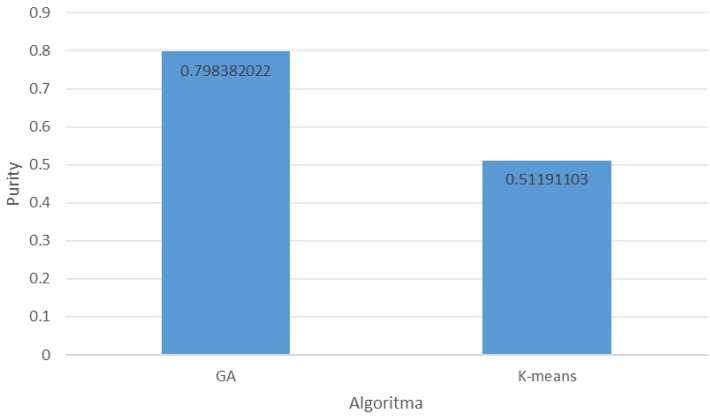
Gambar 5.14: Grafik hubungan algoritma dengan *intracluster similarity*

Berdasarkan grafik pada Gambar 5.14, nilai *intracluster similarity* pada algoritma *K-means* lebih besar 7% daripada algoritma genetika.



Gambar 5.15: Grafik hubungan algoritma dengan banyaknya iterasi

Berdasarkan grafik pada Gambar 5.15, jumlah iterasi dengan menggunakan algoritma *K-means* lebih banyak 150% daripada algoritma genetika. Namun berdasarkan grafik pada Gambar 5.13, waktu yang ditempuh *K-means* lebih sebentar dibandingkan dengan algoritma genetika. Hal ini terjadi karena waktu yang diperlukan untuk menempuh satu iterasi pada *K-means* jauh lebih kecil dibandingkan dengan waktu yang diperlukan untuk menempuh satu iterasi pada algoritma genetika.



Gambar 5.16: Grafik hubungan algoritma dengan nilai *purity*

Berdasarkan grafik pada Gambar 5.16, nilai *purity* algoritma genetika lebih besar 56% dibandingkan dengan nilai *purity* menggunakan algoritma *K-means*.

5.4 Kesimpulan Eksperimen

Berdasarkan eksperimen yang telah dilakukan, dapat diambil beberapa kesimpulan sebagai berikut:

1. Nilai *purity* dan *intracluster similarity* sama sekali tidak berhubungan. Padahal seharusnya semakin besar nilai *intracluster similarity*, maka hasil pengelompokan akan semakin baik. Apabila hasil pengelompokan semakin baik, maka nilai *purity* seharusnya semakin besar pula. Berdasarkan hasil eksperimen, nilai *purity* lebih akurat dalam menentukan baik atau buruknya hasil suatu pengelompokan karena menggunakan label yang telah ada sebelumnya untuk

mengukur kemurnian hasil pengelompokan. Dapat disimpulkan bahwa *intracluster similarity* kurang merepresentasikan seberapa baik suatu hasil pengelompokan.

2. Algoritma genetika lebih baik 56% menurut nilai *purity* (Gambar 5.16). Namun, kekurangan dari algoritma genetika adalah waktu pemrosesan yang jauh lebih lama dibandingkan dengan algoritma *K-means*.

3. Berdasarkan hasil eksperimen dengan menggunakan algoritma *K-means* (Tabel B.7 pada Lampiran B), algoritma *K-means* memang seringkali terjebak pada *local optimum*. Hal ini dibuktikan dengan nilai *purity* yang memiliki perbedaan cukup jauh antara satu dengan yang lainnya. Hasil dengan nilai *purity* yang cukup kecil (misalkan pada baris ke-3 Tabel B.7 dengan nilai *purity* 0.252080274 atau pada baris ke-10 Tabel B.7 dengan nilai *purity* 0.248153619) merupakan contoh *local optimum* yang dialami oleh *K-means*.

4. Berdasarkan hasil eksperimen menggunakan kasus uji 1 (Tabel B.1 pada Lampiran B). Nilai *purity* pada hasil percobaan ini lebih stabil (memiliki rentang nilai yang kecil) dibandingkan dengan menggunakan algoritma *K-means*. Dari lima kali pengujian, tidak ada satupun yang memiliki nilai *purity* jauh dari rata-rata sehingga dapat disimpulkan bahwa algoritma genetika terbukti lebih baik dalam mengatasi *local optimum*.

BAB 6

KESIMPULAN DAN SARAN

6.1 Kesimpulan

Kesimpulan yang dapat diambil dari penelitian ini adalah sebagai berikut:

1. Algoritma genetika dapat digunakan dalam pengelompokan dokumen. Namun, diperlukan beberapa adaptasi terhadap komponen-komponen dalam algoritma genetika sebelum dapat digunakan untuk mengelompokkan dokumen. Adaptasi perlu dilakukan terhadap representasi kromosom, fungsi *fitness*, seleksi, persilangan, dan mutasi yang telah dijelaskan pada Bab 3.

2. Berdasarkan hasil pengujian, *intracluster similarity* kurang merepresentasikan seberapa baik suatu hasil pengelompokan.

3. Nilai *purity* dari hasil pengelompokan menggunakan algoritma genetika lebih baik 56% dibandingkan dengan menggunakan agloritma K-means berdasarkan hasil eksperimen. Hal ini terjadi karena algoritma genetika dapat dengan lebih baik mengatasi *local optimum* dibandingkan dengan algoritma *K-means*. Namun dari segi waktu, algoritma genetika membutuhkan waktu 4365% lebih lama dibandingkan dengan algoritma *K-means*. Hal ini disebabkan oleh proses komputasi yang dilakukan pada algoritma genetika jauh lebih banyak dan kompleks dibandingkan dengan algoritma *K-means*.

4. Representasi kromosom yang kurang tepat juga menjadi alasan algoritma genetika berjalan dengan lambat. Mulai generasi kedua, *centroid* yang menyusun kromosom akan memiliki dimensi yang sangat besar (sesuai dengan banyaknya *term* berbeda yang ada pada *lexicon*). *Centroid* dengan dimensi yang besar akan memperlambat proses perhitungan *fitness*.

6.2 Saran

Saran dari penulis untuk peneliti selanjutnya agar dapat mengembangkan penelitian ini adalah sebagai berikut:

1. Mencari suatu metrik yang dapat mengukur seberapa baik suatu hasil pengelompokan yang dapat menggantikan *intracluster similarity*.

2. Menggunakan representasi kromosom yang lain sehingga dapat menjaga agar dimensi dari *centroid* tidak terlalu besar. Hal ini dapat dilakukan dengan mengubah representasi kromosom menjadi dokumen dan keanggotaannya dalam *cluster*.

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] Srinivas, M. dan Patnaik, L. M. (1994) Genetic algorithms: A survey. *computer*, **27**, 17–26.

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

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

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

[12] 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: FXMLDocument.fxml

```
<?xml version="1.0" encoding="UTF-8"?>

<?import javafx.scene.control.Button?>
<?import javafx.scene.control.ChoiceBox?>
<?import javafx.scene.control.Label?>
<?import javafx.scene.control.ProgressBar?>
<?import javafx.scene.control.Spinner?>
<?import javafx.scene.control.Tab?>
<?import javafx.scene.control.TabPane?>
<?import javafx.scene.control.TextField?>
<?import javafx.scene.layout.AnchorPane?>
<?import javafx.scene.text.Font?>

<AnchorPane id="AnchorPane" prefHeight="433.0" prefWidth="665.0" xmlns="http://javafx.com/javafx/11.0.1" xmlns:fx="http://javafx.com/fxml/1" fx:controller="ga.clustering.gui.FXMLDocumentController">
    <children>
      <Label layoutX="30.0" layoutY="23.0" text="Direktori Dokumen: " />
      <TextField fx:id="textFieldDokumen" editable="false" layoutX="176.0" layoutY="19.0" prefHeight="25.0" prefWidth="354.0" />
      <Button fx:id="buttonDokumen" layoutX="551.0" layoutY="19.0" mnemonicParsing="false" onAction="#chooseDocument" prefHeight="25.0" prefWidth="59.0" text="Pilih" />
      <Label layoutX="30.0" layoutY="60.0" text="Direktori Hasil:" />
      <TextField fx:id="textFieldHasil" editable="false" layoutX="176.0" layoutY="56.0" prefHeight="25.0" prefWidth="354.0" />
      <Button fx:id="buttonHasil" layoutX="551.0" layoutY="56.0" mnemonicParsing="false" onAction="#chooseResult" prefHeight="25.0" prefWidth="59.0" text="Pilih" />
      <TabPane layoutY="90.0" prefHeight="343.0" prefWidth="665.0" styleClass="floating" tabClosingPolicy="UNAVAILABLE">
        <tabs>
          <Tab fx:id="tabGA" text="Genetic Algorithm">
               <content>
                  <AnchorPane prefHeight="316.0" prefWidth="665.0">
                     <children>
                        <Label layoutX="30.0" layoutY="10.0" text="Parameter: ">
                           <font>
                              <Font name="System Bold" size="14.0" />
                           </font>
                        </Label>
                        <Spinner fx:id="spinnerCluster" layoutX="199.0" layoutY="48.0" prefHeight="25.0" prefWidth="104.0" />
                        <Label layoutX="51.0" layoutY="51.0" text="Banyaknya Cluster (K)" />
                        <Label layoutX="51.0" layoutY="90.0" text="Banyaknya Populasi (P)" />
                        <Label layoutX="51.0" layoutY="127.0" text="Metode Pembobotan" />
                        <Label layoutX="51.0" layoutY="164.0" text="Probabilitas Mutasi (   )" />
                        <Label layoutX="51.0" layoutY="201.0" text="Maksimum iterasi" />
                        <ProgressBar fx:id="progressBar" layoutX="28.0" layoutY="251.0" prefHeight="28.0" prefWidth="482.0" progress="0.0" />
                        <Button fx:id="buttonMulai" layoutX="518.0" layoutY="245.0" mnemonicParsing="false" onAction="#start" prefHeight="42.0" prefWidth="134.0" text="Mulai">
                           <font>
                              <Font name="System Bold" size="18.0" />
                           </font>
                        </Button>
                        <Label layoutX="338.0" layoutY="52.0" text="Individu Elitisme" />
                        <Label layoutX="338.0" layoutY="91.0" text="Banyaknya Generasi Konvergen" />
                        <Label layoutX="338.0" layoutY="128.0" text="Batas Konvergen" />
                        <Spinner fx:id="spinnerPopulasi" layoutX="199.0" layoutY="88.0" prefHeight="25.0" prefWidth="104.0" />
                        <Spinner fx:id="spinnerMutasi" layoutX="199.0" layoutY="162.0" prefHeight="25.0" prefWidth="104.0" />
                        <Spinner fx:id="spinnerMaxIterasi" layoutX="199.0" layoutY="198.0" prefHeight="25.0" prefWidth="104.0" />
                        <Spinner fx:id="spinnerElitism" layoutX="522.0" layoutY="48.0" prefHeight="25.0" prefWidth="104.0" />
                        <Spinner fx:id="spinnerConvergeGen" layoutX="522.0" layoutY="88.0" prefHeight="25.0" prefWidth="104.0" />
                        <ChoiceBox fx:id="choiceBoxWeighting" layoutX="199.0" layoutY="124.0" prefHeight="25.0" prefWidth="105.0" />
                        <Spinner fx:id="spinnerConvergeLimit" layoutX="522.0" layoutY="124.0" prefHeight="25.0" prefWidth="104.0" />
                        <Label layoutX="159.0" layoutY="164.0" text="u">
                           <font>
                              <Font name="System Italic" size="12.0" />
                           </font>
                        </Label>
                        <Label layoutX="165.0" layoutY="173.0" text="m">
                           <font>
                              <Font size="5.0" />
                           </font>
                        </Label>
                        <Label fx:id="labelProgress" layoutX="30.0" layoutY="282.0" prefHeight="17.0" prefWidth="477.0" />
                     </children>
                  </AnchorPane>
               </content>
```

```
69          </Tab>
70          <Tab fx:id="tabKMeans" text="K-Means">
71            <content>
72              <AnchorPane minHeight="0.0" minWidth="0.0" prefHeight="180.0" prefWidth="200.0">
73                     <children>
74                        <Label fx:id="KMLabelProgress" layoutX="30.0" layoutY="282.0" prefHeight="17.0" prefWidth="477.0" />
75                        <ProgressBar fx:id="KMprogressBar" layoutX="28.0" layoutY="251.0" prefHeight="28.0" prefWidth="480.0"
                               progress="0.0" />
76                        <Button fx:id="KMbuttonMulai" layoutX="518.0" layoutY="245.0" mnemonicParsing="false" onAction="#KMStart"
                               prefHeight="42.0" prefWidth="134.0" text="Mulai">
77                           <font>
78                              <Font name="System Bold" size="18.0" />
79                           </font>
80                        </Button>
81                        <Spinner fx:id="KMspinnerMaxIterasi" layoutX="200.0" layoutY="123.0" prefHeight="25.0" prefWidth="104.0" /
                              >
82                        <Label layoutX="30.0" layoutY="10.0" text="Parameter: ">
83                           <font>
84                              <Font name="System Bold" size="14.0" />
85                           </font>
86                        </Label>
87                        <Spinner fx:id="KMspinnerCluster" layoutX="199.0" layoutY="48.0" prefHeight="25.0" prefWidth="104.0" />
88                        <Label layoutX="51.0" layoutY="51.0" text="Banyaknya Cluster (K)" />
89                        <Label layoutX="51.0" layoutY="90.0" text="Metode Pembobotan" />
90                        <Label layoutX="51.0" layoutY="127.0" text="Maksimum iterasi" />
91                        <ChoiceBox fx:id="KMchoiceBoxWeighting" layoutX="199.0" layoutY="86.0" prefHeight="25.0" prefWidth="105.0"
                               />
92                     </children></AnchorPane>
93            </content>
94          </Tab>
95        </tabs>
96      </TabPane>
97    </children>
98 </AnchorPane>
```

Listing A.2: FXMLDocumentController.java

```
1  /*
2   * To change this license header, choose License Headers in Project Properties.
3   * To change this template file, choose Tools | Templates
4   * and open the template in the editor.
5   */
6  package ga.clustering.gui;
7
8  import ga.clustering.gui.GA.GAClusterer;
9  import ga.clustering.gui.IR.Document;
10 import ga.clustering.gui.KMeans.KMeans;
11 import java.io.BufferedWriter;
12 import java.io.File;
13 import java.io.FileWriter;
14 import java.net.URL;
15 import java.text.DateFormat;
16 import java.text.SimpleDateFormat;
17 import java.util.ArrayList;
18 import java.util.Date;
19 import java.util.HashMap;
20 import java.util.List;
21 import java.util.Map.Entry;
22 import java.util.ResourceBundle;
23 import javafx.application.Platform;
24 import javafx.collections.FXCollections;
25 import javafx.collections.ObservableList;
26 import javafx.concurrent.Task;
27 import javafx.fxml.FXML;
28 import javafx.fxml.Initializable;
29 import javafx.scene.control.Alert;
30 import javafx.scene.control.Alert.AlertType;
31 import javafx.scene.control.Button;
32 import javafx.scene.control.ChoiceBox;
33 import javafx.scene.control.Label;
34 import javafx.scene.control.ProgressBar;
35 import javafx.scene.control.Skin;
36 import javafx.scene.control.Spinner;
37 import javafx.scene.control.SpinnerValueFactory;
38 import javafx.scene.control.Tab;
39 import javafx.scene.control.TextField;
40 import javafx.stage.DirectoryChooser;
41
42 /**
43  *
44  * @author CorneliusDavid
45  */
46 public class FXMLDocumentController implements Initializable {
47
48     @FXML
49     private TextField textFieldDokumen;
50
51     @FXML
52     private Button buttonDokumen;
53
54     @FXML
55     private Spinner spinnerCluster;
56
57     @FXML
58     private Spinner spinnerPopulasi;
59
60     @FXML
61     private Spinner spinnerMutasi;
```

```
62
63      @FXML
64      private Spinner spinnerMaxIterasi;
65
66      @FXML
67      private Spinner spinnerElitism;
68
69      @FXML
70      private Spinner spinnerConvergeGen;
71
72      @FXML
73      private ChoiceBox choiceBoxWeighting;
74
75      @FXML
76      private Spinner spinnerConvergeLimit;
77
78      @FXML
79      private ProgressBar progressBar;
80
81      @FXML
82      private Button buttonMulai;
83
84      @FXML
85      private TextField textFieldHasil;
86
87      @FXML
88      private Button buttonHasil;
89
90      @FXML
91      private Tab tabGA;
92
93      @FXML
94      private Tab tabKMeans;
95
96      @FXML
97      private Label labelProgress;
98
99      //KMEANS
100     @FXML
101     private Spinner KMspinnerCluster;
102
103     @FXML
104     private ChoiceBox KMchoiceBoxWeighting;
105
106     @FXML
107     private Spinner KMspinnerMaxIterasi;
108
109     @FXML
110     private ProgressBar KMprogressBar;
111
112     @FXML
113     private Button KMbuttonMulai;
114
115     @FXML
116     private Label KMLabelProgress;
117
118     private Thread thread;
119
120     private long runningTime;
121
122     private Task task;
123
124     private int curIt;
125
126     @Override
127     public void initialize(URL url, ResourceBundle rb) {
128         //hasil
129         File initial=new File("res\\");
130         this.textFieldHasil.setText(initial.getAbsolutePath());
131
132         //cluster
133         SpinnerValueFactory<Integer> clusterValueFactory =new SpinnerValueFactory.IntegerSpinnerValueFactory(1,Integer.MAX_VALUE
                ,5);
134         this.spinnerCluster.setValueFactory(clusterValueFactory);
135
136         //populasi
137         SpinnerValueFactory<Integer> populationValueFactory =new SpinnerValueFactory.IntegerSpinnerValueFactory(1,Integer.
                MAX_VALUE,100);
138         this.spinnerPopulasi.setValueFactory(populationValueFactory);
139
140         //mutasi
141         SpinnerValueFactory<Double> mutationValueFactory=new SpinnerValueFactory.DoubleSpinnerValueFactory(0, 1.0, 0.05, 0.01);
142         this.spinnerMutasi.setValueFactory(mutationValueFactory);
143
144         //max iterasi
145         SpinnerValueFactory<Integer> iterationValueFactory =new SpinnerValueFactory.IntegerSpinnerValueFactory(1,Integer.MAX_VALUE
                ,100);
146         this.spinnerMaxIterasi.setValueFactory(iterationValueFactory);
147
148         //weighting
149         this.choiceBoxWeighting.setItems(FXCollections.observableArrayList(
150                 "TF-IDF", "Frekuensi"
151         ));
152         this.choiceBoxWeighting.getSelectionModel().selectFirst();
153
154         //converge boundary
155         ObservableList<String> values=FXCollections.observableArrayList(
156                 "0.0000001","0.000001","0.00001","0.0001","0.001"
157         );
```

```
158         SpinnerValueFactory<String> limitValueFactory=new SpinnerValueFactory.ListSpinnerValueFactory<>(values);
159         limitValueFactory.setValue("0.00001");
160         this.spinnerConvergeLimit.setValueFactory(limitValueFactory);
161 
162         //elitism
163         SpinnerValueFactory<Integer> elitismValueFactory =new SpinnerValueFactory.IntegerSpinnerValueFactory(0,Integer.MAX_VALUE
                ,1);
164         this.spinnerElitism.setValueFactory(elitismValueFactory);
165         attachWarning();
166 
167         //converge Generation
168         SpinnerValueFactory<Integer> convergeGenValueFactory =new SpinnerValueFactory.IntegerSpinnerValueFactory(2,Integer.
                MAX_VALUE,3);
169         this.spinnerConvergeGen.setValueFactory(convergeGenValueFactory);
170 
171         //KMEANS
172         //cluster
173         SpinnerValueFactory<Integer> KMclusterValueFactory =new SpinnerValueFactory.IntegerSpinnerValueFactory(1,Integer.MAX_VALUE
                ,5);
174         this.KMspinnerCluster.setValueFactory(KMclusterValueFactory);
175 
176         //weighting
177         this.KMchoiceBoxWeighting.setItems(FXCollections.observableArrayList(
178                 "TF-IDF", "Frekuensi"
179         ));
180         this.KMchoiceBoxWeighting.getSelectionModel().selectFirst();
181 
182         //max iterasi
183         SpinnerValueFactory<Integer> KMiterationValueFactory =new SpinnerValueFactory.IntegerSpinnerValueFactory(1,Integer.
                MAX_VALUE,100);
184         this.KMspinnerMaxIterasi.setValueFactory(KMiterationValueFactory);
185     }
186 
187     private void attachWarning(){
188         this.spinnerElitism.valueProperty().addListener((observable, oldValue, newValue) -> warningPopulation(observable, oldValue
                , newValue));
189         this.spinnerPopulasi.valueProperty().addListener((observable, oldValue, newValue) -> warningPopulation(observable,
                oldValue, newValue));
190         this.spinnerConvergeGen.valueProperty().addListener((observable, oldValue, newValue) -> warningIteration(observable,
                oldValue, newValue));
191         this.spinnerMaxIterasi.valueProperty().addListener((observable, oldValue, newValue) -> warningIteration(observable,
                oldValue, newValue));
192     }
193 
194     private void warningPopulation(Object observable, Object oldValue, Object newValue){
195         if((int)this.spinnerElitism.valueProperty().getValue()>(int)this.spinnerPopulasi.valueProperty().getValue()){
196             this.spinnerElitism.getValueFactory().setValue(oldValue);
197             this.spinnerPopulasi.getValueFactory().setValue(oldValue);
198             try{
199                 Skin<?> skin = spinnerElitism.getSkin();
200                 Object behavior = skin.getClass().getMethod("getBehavior").invoke(skin);
201                 behavior.getClass().getMethod("stopSpinning").invoke(behavior);
202 
203                 skin = spinnerPopulasi.getSkin();
204                 behavior = skin.getClass().getMethod("getBehavior").invoke(skin);
205                 behavior.getClass().getMethod("stopSpinning").invoke(behavior);
206             }catch(Exception e){}
207             Alert alert = new Alert(AlertType.WARNING);
208             alert.setTitle("Peringatan!");
209             alert.setHeaderText("Nilai tidak valid");
210             alert.setContentText("Nilai Individu Elitisme tidak bisa lebih besar dari Banyaknya Populasi");
211 
212             alert.showAndWait();
213         }
214     }
215 
216     private void warningIteration(Object observable, Object oldValue, Object newValue){
217         if((int)this.spinnerConvergeGen.valueProperty().getValue()>(int)this.spinnerMaxIterasi.valueProperty().getValue()){
218             this.spinnerConvergeGen.getValueFactory().setValue(oldValue);
219             this.spinnerMaxIterasi.getValueFactory().setValue(oldValue);
220             try{
221                 Skin<?> skin = spinnerConvergeGen.getSkin();
222                 Object behavior = skin.getClass().getMethod("getBehavior").invoke(skin);
223                 behavior.getClass().getMethod("stopSpinning").invoke(behavior);
224 
225                 skin = spinnerMaxIterasi.getSkin();
226                 behavior = skin.getClass().getMethod("getBehavior").invoke(skin);
227                 behavior.getClass().getMethod("stopSpinning").invoke(behavior);
228             }catch(Exception e){}
229             Alert alert = new Alert(AlertType.WARNING);
230             alert.setTitle("Peringatan!");
231             alert.setHeaderText("Nilai tidak valid");
232             alert.setContentText("Banyaknya Generasi Konvergen tidak bisa lebih besar dari Maksimum Iterasi");
233 
234             alert.showAndWait();
235         }
236     }
237 
238     public void chooseDocument(){
239         DirectoryChooser directoryChooser=new DirectoryChooser();
240         File selected=directoryChooser.showDialog(this.buttonDokumen.getScene().getWindow());
241         if(selected!=null){
242             this.textFieldDokumen.setText(selected.getAbsolutePath());
243         }
244     }
245 
246     public void chooseResult(){
247         DirectoryChooser directoryChooser=new DirectoryChooser();
248         File initial=new File("res\\");
```

```
249         if(!initial.exists()){
250             initial.mkdirs();
251         }
252         directoryChooser.setInitialDirectory(initial);
253         File selected=directoryChooser.showDialog(this.buttonHasil.getScene().getWindow());
254         if(selected!=null){
255             this.textFieldHasil.setText(selected.getAbsolutePath());
256         }
257     }
258 
259     private void reset(){
260         this.buttonDokumen.disableProperty().set(false);
261         this.spinnerCluster.disableProperty().set(false);
262         this.spinnerConvergeGen.disableProperty().set(false);
263         this.spinnerConvergeLimit.disableProperty().set(false);
264         this.spinnerElitism.disableProperty().set(false);
265         this.spinnerMaxIterasi.disableProperty().set(false);
266         this.spinnerMutasi.disableProperty().set(false);
267         this.spinnerPopulasi.disableProperty().set(false);
268         this.choiceBoxWeighting.disableProperty().set(false);
269         this.textFieldDokumen.disableProperty().set(false);
270         this.textFieldHasil.disableProperty().set(false);
271         this.buttonHasil.disableProperty().set(false);
272         this.buttonMulai.setText("Mulai");
273         this.tabGA.disableProperty().set(false);
274         this.tabKMeans.disableProperty().set(false);
275 
276         this.curIt=0;
277         this.labelProgress.textProperty().unbind();
278         this.labelProgress.setText("");
279         this.runningTime=0;
280         Platform.runLater(new Runnable() {
281             @Override
282             public void run() {
283                 progressBar.progressProperty().unbind();
284                 progressBar.setProgress(0);
285             }
286         });
287         GAClusterer.getInstance().reset();
288     }
289 
290     private void KMReset(){
291         this.KMchoiceBoxWeighting.disableProperty().set(false);
292         this.KMspinnerCluster.disableProperty().set(false);
293         this.KMspinnerMaxIterasi.disableProperty().set(false);
294         this.tabGA.disableProperty().set(false);
295 
296         this.KMbuttonMulai.setText("Mulai");
297 
298         this.curIt=0;
299         this.KMLabelProgress.textProperty().unbind();
300         this.KMLabelProgress.setText("");
301         this.runningTime=0;
302         KMeans.getInstance().reset();
303 
304         Platform.runLater(new Runnable() {
305             @Override
306             public void run() {
307                 KMprogressBar.progressProperty().unbind();
308                 KMprogressBar.setProgress(0);
309             }
310         });
311     }
312 
313     public void start() throws Exception{
314         if(this.buttonMulai.getText().equalsIgnoreCase("reset")){
315             reset();
316             return;
317         }
318 
319         if(this.buttonMulai.getText().equalsIgnoreCase("berhenti")){
320             task.cancel();
321             return;
322         }
323 
324         if(this.textFieldDokumen.getText().length()==0){
325             Alert alert = new Alert(AlertType.WARNING);
326             alert.setTitle("Peringatan!");
327             alert.setHeaderText("Nilai_tidak_valid");
328             alert.setContentText("Direktori_untuk_dokumen_belum_dipilih");
329 
330             alert.showAndWait();
331             return;
332         }
333         String filepath=this.textFieldDokumen.getText();
334         int K=(int)this.spinnerCluster.getValue();
335         int P=(int)this.spinnerPopulasi.getValue();
336         int weightMethod=this.choiceBoxWeighting.getSelectionModel().getSelectedIndex(); //0=TF-IDF, 1=Frekuensi
337         double mu_m=Double.parseDouble(this.spinnerMutasi.getValue().toString());
338         int maxIt=(int)this.spinnerMaxIterasi.getValue();
339         int elitismCount=(int)this.spinnerElitism.getValue();
340         int convergeGen=(int)this.spinnerConvergeGen.getValue();
341         double convergeEpsilon=Double.parseDouble(this.spinnerConvergeLimit.getValue().toString());
342 
343         curIt=1;
344         Params.getInstance().insertParam(filepath, K, P, weightMethod, mu_m, maxIt, elitismCount, convergeGen, convergeEpsilon);
345         task=new Task() {
346             @Override
347             protected Object call() throws Exception {
```

```
348                 GAClusterer.getInstance().progressProperty().addListener((obs, oldProgress, newProgress) -> {
349                     if(newProgress.doubleValue()==1){
350                         updateMessage(String.format("Generasi␣%d",++curIt));
351                     }
352                     updateProgress(newProgress.doubleValue(), 1);
353                 });
354                 updateMessage("Inisialisasi...");
355                 GAClusterer.getInstance().initialize();
356                 updateMessage("Generasi␣1");
357                 updateProgress(0, 100);
358                 GAClusterer.getInstance().cluster();
359                 return true;
360             }
361         };
362         this.progressBar.progressProperty().bind(task.progressProperty());
363         this.labelProgress.textProperty().bind(task.messageProperty());
364 
365         //disable all
366         this.buttonDokumen.disableProperty().set(true);
367         this.spinnerCluster.disableProperty().set(true);
368         this.spinnerConvergeGen.disableProperty().set(true);
369         this.spinnerConvergeLimit.disableProperty().set(true);
370         this.spinnerElitism.disableProperty().set(true);
371         this.spinnerMaxIterasi.disableProperty().set(true);
372         this.spinnerMutasi.disableProperty().set(true);
373         this.spinnerPopulasi.disableProperty().set(true);
374         this.choiceBoxWeighting.disableProperty().set(true);
375         this.textFieldDokumen.disableProperty().set(true);
376         this.textFieldHasil.disableProperty().set(true);
377         this.buttonHasil.disableProperty().set(true);
378         this.buttonMulai.setText("Berhenti");
379         this.tabKMeans.disableProperty().set(true);
380 
381         thread=new Thread(task);
382         long currentTime=System.currentTimeMillis();
383 
384         thread.start();
385         task.setOnSucceeded((event) -> {
386             this.buttonMulai.setText("Reset");
387             this.runningTime=System.currentTimeMillis()-currentTime;
388             writeToFile("GA",GAClusterer.getInstance().getSolution().getClusteringResult(), GAClusterer.getInstance().getSolution
                    ().getFitness());
389             this.labelProgress.textProperty().unbind();
390             this.labelProgress.setText("Selesai");
391         });
392 
393         task.setOnCancelled((event) -> {
394             task.cancel();
395             GAClusterer.getInstance().setIsRunning(false);
396             this.labelProgress.textProperty().unbind();
397             this.labelProgress.setText("");
398             reset();
399         });
400     }
401 
402 
403     public void KMStart(){
404         if(this.KMbuttonMulai.getText().equalsIgnoreCase("reset")){
405             KMReset();
406             return;
407         }
408 
409         if(this.KMbuttonMulai.getText().equalsIgnoreCase("berhenti")){
410             task.cancel();
411             return;
412         }
413 
414         if(this.textFieldDokumen.getText().length()==0){
415             Alert alert = new Alert(AlertType.WARNING);
416             alert.setTitle("Peringatan!");
417             alert.setHeaderText("Nilai␣tidak␣valid");
418             alert.setContentText("Direktori␣untuk␣dokumen␣belum␣dipilih");
419 
420             alert.showAndWait();
421             return;
422         }
423         String filepath=this.textFieldDokumen.getText();
424         int K=(int)this.KMspinnerCluster.getValue();
425         int weightMethod=this.KMchoiceBoxWeighting.getSelectionModel().getSelectedIndex(); //0=TF-IDF, 1=Frekuensi
426         int maxIt=(int)this.KMspinnerMaxIterasi.getValue();
427 
428         Params.getInstance().insertParam(filepath, K, -1, weightMethod, -1, maxIt, -1, -1, -1);
429         curIt=1;
430         task=new Task() {
431             @Override
432             protected Object call() throws Exception {
433                 KMeans.getInstance().progressProperty().addListener((obs, oldProgress, newProgress) -> {
434                     if(newProgress.doubleValue()==1){
435                         updateMessage(String.format("Iterasi␣%d",++curIt));
436                     }
437                     updateProgress(newProgress.doubleValue(), 1);
438                 });
439                 updateMessage("Iterasi␣1");
440                 KMeans.getInstance().cluster();
441                 return true;
442             }
443         };
444         this.KMprogressBar.progressProperty().bind(task.progressProperty());
445         this.KMLabelProgress.textProperty().bind(task.messageProperty());
```

```
446
447         this.KMchoiceBoxWeighting.disableProperty().set(true);
448         this.KMspinnerCluster.disableProperty().set(true);
449         this.KMspinnerMaxIterasi.disableProperty().set(true);
450         this.tabGA.disableProperty().set(true);
451         this.KMbuttonMulai.setText("Berhenti");
452 
453         long currentTime = System.currentTimeMillis();
454         task.setOnSucceeded((event) -> {
455             this.KMbuttonMulai.setText("Reset");
456             this.runningTime=System.currentTimeMillis()-currentTime;
457             writeToFile("KMeans",KMeans.getInstance().getSolution(), KMeans.getInstance().getSolutionIntracluster());
458             this.KMLabelProgress.textProperty().unbind();
459             this.KMLabelProgress.setText("Selesai");
460         });
461 
462         task.setOnCancelled((event) -> {
463             task.cancel();
464             KMeans.getInstance().setIsRunning(false);
465             this.KMLabelProgress.textProperty().unbind();
466             this.KMLabelProgress.setText("");
467             KMReset();
468         });
469         thread=new Thread(task);
470         thread.start();
471     }
472 
473     private String timeFormatter(long timeMillis){
474         int timeSecond=(int)timeMillis/1000;
475         int hour=timeSecond/3600;
476         timeSecond=timeSecond%3600;
477         int minutes=timeSecond/60;
478         timeSecond=timeSecond%60;
479         int second=timeSecond;
480 
481         String result="";
482         result+=hour>0?hour+"␣jam␣":"";
483         result+=minutes>0?minutes+"␣menit␣":"";
484         result+=second+"␣detik";
485         return result;
486     }
487 
488     private void writeToFile(String mode, HashMap<Document, Integer> clusteringResult, double fitness){
489         try{
490             System.out.println("writing...");
491             String filepath=Params.getInstance().getFilepath();
492             List<Document>[] result=new ArrayList[Params.getInstance().getK()];
493             for (int i = 0; i < result.length; i++) {
494                 result[i]=new ArrayList<>();
495             }
496             for(Entry<Document,Integer> entry:clusteringResult.entrySet()){
497                 result[entry.getValue()].add(entry.getKey());
498             }
499             int maxLength=0;
500             for (int i = 0; i < result.length; i++) {
501                 maxLength=Math.max(maxLength, result[i].size());
502             }
503             DateFormat dateFormat = new SimpleDateFormat("yyyy.MM.dd␣HH␣mm␣ss");
504             Date date = new Date();
505             String pathHasil=textFieldHasil.getText();
506             String filename=pathHasil+"\\"+mode+"-"+dateFormat.format(date)+".csv";
507             BufferedWriter bw = new BufferedWriter(new FileWriter(filename));
508 
509             //write header
510             bw.write("Direktori␣Dokumen:,"+filepath+"\r\n␣\r\n");
511             bw.write("Parameter:␣\r\n");
512             if(Params.getInstance().getK()!=-1)bw.write("Banyaknya␣Cluster,"+Params.getInstance().getK()+"\r\n");
513             if(Params.getInstance().getP()!=-1)bw.write("Banyaknya␣Populasi,"+Params.getInstance().getP()+"\r\n");
514             if(Params.getInstance().getWeightingMethod()!=-1)bw.write("Metode␣Pembobotan,"+(Params.getInstance().
                    getWeightingMethod()==0?"TF-IDF":"Frekuensi")+"\r\n");
515             if(Params.getInstance().getMu␣m()!=-1)bw.write("Probabilitas␣Mutasi,"+Params.getInstance().getMu␣m()+"\r\n");
516             if(Params.getInstance().getMaxIt()!=-1)bw.write("Maksimum␣Iterasi,"+Params.getInstance().getMaxIt()+"\r\n");
517             if(Params.getInstance().getElitismCount()!=-1)bw.write("Individu␣Elitisme,"+Params.getInstance().getElitismCount()+"\r
                    \n");
518             if(Params.getInstance().getConvergeGen()!=-1)bw.write("Banyaknya␣Generasi␣Konvergen,"+Params.getInstance().
                    getConvergeGen()+"\r\n");
519             if(Params.getInstance().getConvergeEpsilon()!=-1)bw.write("Batas␣Konvergen,"+Params.getInstance().getConvergeEpsilon()
                    +"\r\n");
520 
521             //write hasil
522             bw.write("\r\nHasil:␣\r\n");
523             bw.write("Waktu,"+timeFormatter(runningTime)+"\r\n");
524             bw.write("Intracluster,"+fitness+"\r\n");
525             bw.write("Banyak␣Iterasi,"+(curIt-1)+"\r\n␣\r\n");
526 
527             //write hasil clustering
528             bw.write("Hasil␣Clustering:␣\r\n");
529             for (int i = 0; i < result.length; i++) {
530                 bw.write("C"+(i+1)+",");
531             }
532             bw.write("\r\n");
533             for (int i = 0; i < maxLength; i++) {
534                 for (int j = 0; j < result.length; j++) {
535                     if(i<result[j].size()){
536                         String name=result[j].get(i).getDocName().substring(filepath.length());
537                         if(name.charAt(0)=='\\'){
538                             name=name.substring(1);
539                         }
540                         bw.write(name);
```

```
                    }
                    bw.write(",");
                }
                bw.write("\r\n");
            }
            bw.close();
        }catch(Exception e){}
        System.out.println("written");
    }
}
```

Listing A.3: Params.java

```
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package ga.clustering.gui;

/**
 *
 * @author Cornelius David
 */
public class Params {
    private static Params instance;
    private String filepath;
    private int K;
    private int P;
    private int weightingMethod; //0=TF-IDF, 1=Frekuensi
    private double mu_m;
    private int maxIt;
    private int elitismCount;
    private int convergeGen;
    private double convergeEpsilon;

    private Params(){

    }

    public static Params getInstance(){
        if(instance==null){
            instance=new Params();
        }
        return instance;
    }

    public void insertParam(String filepath, int K, int P, int weightMethod, double mu_m, int maxIt, int elitismCount, int
          convergeGen, double convergeEpsilon){
        this.K=K;
        this.P=P;
        this.weightingMethod=weightMethod;
        this.mu_m=mu_m;
        this.maxIt=maxIt;
        this.elitismCount=elitismCount;
        this.convergeGen=convergeGen;
        this.convergeEpsilon=convergeEpsilon;
        this.filepath=filepath;
    }

    public int getK() {
        return K;
    }

    public int getP() {
        return P;
    }

    public int getWeightingMethod() {
        return weightingMethod;
    }

    public double getMu_m() {
        return mu_m;
    }

    public int getMaxIt() {
        return maxIt;
    }

    public int getElitismCount() {
        return elitismCount;
    }

    public int getConvergeGen() {
        return convergeGen;
    }

    public double getConvergeEpsilon() {
        return convergeEpsilon;
    }

    public String getFilepath() {
        return filepath;
    }
}
```

Listing A.4: Chromosome.java

```java
package ga.clustering.gui.GA;



import ga.clustering.gui.Params;
import ga.clustering.gui.IR.Document;
import ga.clustering.gui.IR.Lexicon;
import ga.clustering.gui.IR.Vector;
import java.util.HashMap;
import java.util.LinkedList;
import java.util.List;
import java.util.Random;

/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */

/**
 *
 * @author CorneliusDavid
 */
public class Chromosome implements Comparable<Chromosome>{
    private List<Gene> genes;
    private Random rand;
    private double fitnessValue;
    private HashMap<Document, Integer> clusteringResult;

    public Chromosome() {
        this.genes = new LinkedList<>();
        this.rand = new Random();
        this.fitnessValue=-1;
        this.clusteringResult=new HashMap<>();
    }

    public void addGene(Gene gene){
        this.genes.add(gene);
    }

    public void mutate(){
        if(rand.nextDouble()<Params.getInstance().getMu_m()){
            int mutatedGeneIdx=rand.nextInt(genes.size());
            genes.get(mutatedGeneIdx).mutate();
//            int mutatedGeneIdx1=rand.nextInt(genes.size());
//            int mutatedGeneIdx2=(mutatedGeneIdx1+rand.nextInt(genes.size()))%genes.size();
//            Vector diff=new Vector(new HashMap<>());
//            for(String t:genes.get(mutatedGeneIdx2).getValue().getKeySet()){
//                Vector tmp1=genes.get(mutatedGeneIdx1).getValue();
//                Vector tmp2=genes.get(mutatedGeneIdx2).getValue();
//                diff.setWeight(t, tmp2.getWeight(t)-tmp1.getWeight(t));
//            }
//            Random r=new Random();
//            double factor=r.nextDouble()*(r.nextBoolean()?-1:1);
//            diff.getKeySet().forEach((t) -> {
//                diff.setWeight(t, diff.getWeight(t)*factor);
//            });
//            diff.getKeySet().forEach((t)->{
//                double newVal=diff.getWeight(t)+genes.get(mutatedGeneIdx1).getValue().getWeight(t);
//                newVal=newVal<0?0:newVal;
//                genes.get(mutatedGeneIdx1).getValue().setWeight(t, newVal);
//            });
        }
    }

    public Chromosome crossover(Chromosome otherChromosome){
        int breakPoint=rand.nextInt(genes.size());
        Chromosome result=new Chromosome();
        for (int i = 0; i < breakPoint; i++) {
            result.addGene(new Gene(this.genes.get(i)));
        }

        for (int i = breakPoint; i < this.genes.size(); i++) {
            result.addGene(new Gene(otherChromosome.genes.get(i)));
        }

        return result;
    }

    private void determineCluster(List<Document> docs){
        for (int i = 0; i < docs.size(); i++) {
            int cluster=0;
            double similarity=Double.MIN_VALUE;
            for (int j = 0; j < genes.size(); j++) {
                double temp=docs.get(i).getVector().calculateSimilarity(genes.get(j).getValue());
                if(temp>similarity){
                    similarity=temp;
                    cluster=j;
                }
            }
            this.clusteringResult.put(docs.get(i), cluster);
        }
    }

    public double computeFitness(){
//        if(fitnessValue!=-1)return fitnessValue;
        List<Document> docs=GAClusterer.getInstance().getAllDocs();
```

```
98          HashMap<String, Double> sumOfCentroid[]=new HashMap[genes.size()];
99          for (int i = 0; i < genes.size(); i++) {
100             sumOfCentroid[i]=new HashMap<>();
101         }
102         int pointCount[]=new int[genes.size()];
103
104         //assign dokumen ke cluster
105         long cur=System.currentTimeMillis();
106         determineCluster(docs);
107 //        System.out.println("assignment: "+(System.currentTimeMillis()-cur)/1000.0);
108         cur=System.currentTimeMillis();
109
110         //hitung centroid baru
111         for (int i = 0; i < docs.size(); i++) {
112             int clusterCode=clusteringResult.get(docs.get(i));
113             for(String term:docs.get(i).getVector().getKeySet()){
114                 double curValue=sumOfCentroid[clusterCode].containsKey(term)?sumOfCentroid[clusterCode].get(term):0;
115                 curValue+=docs.get(i).getVector().getWeight(term);
116                 if(curValue!=0){
117                     sumOfCentroid[clusterCode].put(term, curValue);
118                 }
119             }
120             pointCount[clusterCode]++;
121         }
122
123 //        System.out.println("compute new centroid: "+(System.currentTimeMillis()-cur)/1000.0);
124         cur=System.currentTimeMillis();
125
126 //        HashMap copy[]=new HashMap[genes.size()];
127
128         for (int i = 0; i < genes.size(); i++) {
129 //            copy[i]=new HashMap(genes.get(i).getValue().getTermsWeight());
130             for(String term:Lexicon.getInstance().getAllTermList()){
131                 double nextValue=sumOfCentroid[i].containsKey(term)?sumOfCentroid[i].get(term):0;
132                 nextValue=pointCount[i]==0.0?0.0:nextValue/pointCount[i];
133                 if(nextValue!=0){
134                     genes.get(i).getValue().setWeight(term, nextValue);
135                 }
136             }
137         }
138
139         //best document
140 //        Document best[]=new Document[genes.size()];
141 //        double dist[]=new double[genes.size()];
142 //        for (int i = 0; i < dist.length; i++) {
143 //            dist[i]=Double.MIN_VALUE;
144 //        }
145 //
146 //        for (int i = 0; i < docs.size(); i++) {
147 //            int idx=clusteringResult.get(docs.get(i));
148 //            double tempDist=docs.get(i).getVector().calculateSimilarity(genes.get(idx).getValue());
149 //            if(tempDist>dist[idx]){
150 //                dist[idx]=tempDist;
151 //                best[idx]=docs.get(i);
152 //            }
153 //        }
154 //        for (int i = 0; i < best.length; i++) {
155 //            if(best[i]!=null){
156 //                genes.get(i).getValue().setTermsWeight(best[i].getVector().getTermsWeight());
157 //            }else{
158 //                genes.get(i).getValue().setTermsWeight(copy[i]);
159 //            }
160 //
161 ////            if(genes.get(i).getValue().getDimension()>5000){
162 ////                int axssss=0;
163 ////            }
164 //        }
165 ////        System.out.print("count: ");
166 ////        for (int i = 0; i < genes.size(); i++) {
167 ////            if(genes.get(i).getValue().getDimension()>5000){
168 ////                int axssss=0;
169 ////            }
170 ////            System.out.print(genes.get(i).getValue().getDimension()+" ");
171 ////        }
172 ////        System.out.println();
173         this.fitnessValue=-1;
174         return getFitness();
175     }
176
177     public double getFitness(){
178         if(fitnessValue!=-1)return this.fitnessValue;
179         List<Document> docs=GAClusterer.getInstance().getAllDocs();
180         double res=0.0;
181         determineCluster(docs);
182         for (int i = 0; i < docs.size(); i++) {
183             int clusterCode=clusteringResult.get(docs.get(i));
184             double tmp=docs.get(i).getVector().calculateSimilarity(this.genes.get(clusterCode).getValue());
185             res+=tmp;
186         }
187         fitnessValue=res;
188         return res;
189     }
190
191     public List<Gene> getAllGenes() {
192         return genes;
193     }
194
195     public HashMap<Document, Integer> getClusteringResult() {
196         return clusteringResult;
```

```
197     }
198
199     @Override
200     public int compareTo(Chromosome o) {
201         return this.getFitness()>o.getFitness()?-1:this.getFitness()<o.getFitness()?1:0;
202     }
203 }
```

Listing A.5: GAClusterer.java

```
1  package ga.clustering.gui.GA;
2
3  import ga.clustering.gui.Params;
4  import ga.clustering.gui.IR.Document;
5  import ga.clustering.gui.IR.Lexicon;
6  import java.io.File;
7  import java.util.ArrayList;
8  import java.util.Collections;
9  import java.util.LinkedList;
10 import java.util.List;
11 import java.util.PriorityQueue;
12 import java.util.Queue;
13 import java.util.Random;
14 import javafx.beans.property.ReadOnlyDoubleProperty;
15 import javafx.beans.property.ReadOnlyDoubleWrapper;
16
17 /*
18  * To change this license header, choose License Headers in Project Properties.
19  * To change this template file, choose Tools | Templates
20  * and open the template in the editor.
21  */
22 /**
23  *
24  * @author CorneliusDavid
25  */
26 public class GAClusterer{
27
28     private List<Chromosome> population;
29     private List<Document> docs;
30     private static GAClusterer instance;
31     private List<Chromosome> solutionList;
32
33     private boolean isRunning;
34
35     private final ReadOnlyDoubleWrapper progress = new ReadOnlyDoubleWrapper();
36
37     private GAClusterer() {
38         population = new LinkedList<>();
39         docs = new LinkedList<>();
40         solutionList = new LinkedList<>();
41         isRunning=true;
42     }
43
44     public void setIsRunning(boolean isRunning) {
45         this.isRunning = isRunning;
46     }
47
48     public double getProgress() {
49         return progressProperty().get();
50     }
51
52     public ReadOnlyDoubleProperty progressProperty() {
53         return progress ;
54     }
55
56     public static GAClusterer getInstance() {
57         if (instance == null) {
58             instance = new GAClusterer();
59         }
60         return instance;
61     }
62
63     public Chromosome rouletteWheelSelect() {
64         double totalWeight = 0;
65         for (int i = 0; i < population.size(); i++) {
66             totalWeight += population.get(i).getFitness();
67         }
68
69         double selectedValue = (new Random()).nextDouble() * totalWeight;
70
71         for (int i = 0; i < population.size(); i++) {
72             selectedValue -= population.get(i).getFitness();
73             if (selectedValue <= 0) {
74                 return population.get(i);
75             }
76         }
77         return population.get(population.size() - 1);
78     }
79
80     private List<Chromosome> elitism(int numOfInstance) {
81         PriorityQueue<Chromosome> q = new PriorityQueue<>();
82         for (int i = 0; i < population.size(); i++) {
83             if(numOfInstance==0)break;
84             if (i < numOfInstance) {
85                 q.offer(population.get(i));
86             } else {
87                 Chromosome cur = population.get(i);
88                 if (cur.getFitness() > q.peek().getFitness()) {
```

```
                    q.poll();
                    q.offer(cur);
                }
            }
        }
        ArrayList<Chromosome> res=new ArrayList<>(q);
//        System.out.println("elite");
//        for (int i = 0; i < res.size(); i++) {
//            System.out.println(res.get(i).getFitness());
//        }
//        System.out.println("end...");
        return res;
    }

    public List<Chromosome> selection(int elitismCount) {
        long cur = System.currentTimeMillis();
        List<Chromosome> nextGen = elitism(elitismCount);
        System.out.println("elitism: " + (System.currentTimeMillis() - cur) / 1000.0);
        cur = System.currentTimeMillis();
        for (int i = 0; i < population.size() - elitismCount; i++) {
//            long roul = System.currentTimeMillis();
            Chromosome parentA = this.rouletteWheelSelect();
//            System.out.println("single-roulette: "+(System.currentTimeMillis()-roul));
            Chromosome parentB = this.rouletteWheelSelect();
            Chromosome offspring = parentA.crossover(parentB);
//            offspring.computeFitness();
//            HashMap<Document,Integer> tmp=new HashMap<>(offspring.getClusteringResult());
            offspring.mutate();
//            offspring.computeFitness();
//            String debug="";
//            for (int j = 0; j < docs.size(); j++) {
//                if(!Objects.equals(tmp.get(docs.get(j)), offspring.getClusteringResult().get(docs.get(j)))){
//                    debug+=docs.get(j).getDocName().substring(Params.getInstance().getFilepath().length())+"("+tmp.get(docs.get(
 j))+","+offspring.getClusteringResult().get(docs.get(j))+"), ";
//                }
//            }
//            System.out.println(debug);

            nextGen.add(offspring);
            progress.set(0.6+(0.2*i/population.size()));
        }
        System.out.println("roulette: " + (System.currentTimeMillis() - cur) / 1000.0);
        return nextGen;
    }

    public void initialize() {
        String filepath = Params.getInstance().getFilepath();
        int k = Params.getInstance().getK();
        int popSize = Params.getInstance().getP();

        long cur = System.currentTimeMillis();
        //document indexing
        File folder = new File(filepath);
        File[] files = folder.listFiles();
        Queue<File> queue = new LinkedList<>();
        for (File file : files) {
            queue.offer(file);
        }
        LinkedList<File> allFiles = new LinkedList<>();

        while (!queue.isEmpty()) {
            File tmp = queue.poll();
            if (tmp.isDirectory()) {
                File[] tmpFolder = tmp.listFiles();
                for (int i = 0; i < tmpFolder.length; i++) {
                    queue.offer(tmpFolder[i]);
                }
            } else {
                allFiles.add(tmp);
            }
        }

        Lexicon.getInstance().setNumberOfDocument(allFiles.size());

        allFiles.forEach((file) -> {
            docs.add(new Document(file));
        });

        List<Document> list = new LinkedList<>();
        list.addAll(docs);

        System.out.println("Precompute: " + (System.currentTimeMillis() - cur) / 1000.0);
        cur = System.currentTimeMillis();
        //generate initial population
        for (int i = 0; i < popSize; i++) {
            Chromosome temp = new Chromosome();
            Collections.shuffle(list);
            for (int j = 0; j < k; j++) {
                temp.addGene(new Gene(list.get(j).getVector()));
            }
            population.add(temp);
        }

        System.out.println("Init pop: " + (System.currentTimeMillis() - cur) / 1000.0);
    }

    public List<Document> getAllDocs() {
        return docs;
    }
```

```
187
188     public void cluster() {
189         for (int a = 0; a < Params.getInstance().getMaxIt(); a++) {
190             progress.set(0);
191             if(!isRunning){
192                 progress.set(0);
193                 return;
194             }
195             System.out.println("Gen-"+(a+1)+":");
196             int popSize = Params.getInstance().getP();
197             int elitismCount = Params.getInstance().getElitismCount();
198             int convergeGen = Params.getInstance().getConvergeGen();
199             double convergeEpsilon = Params.getInstance().getConvergeEpsilon();
200             long cur = System.currentTimeMillis();
201     //         System.out.println("Gen " + (i + 1));
202             //compute fitness : 0.6
203             double totalTime = 0;
204             for (int i = 0; i < popSize; i++) {
205                 if(!isRunning){
206                     progress.set(0);
207                     return;
208                 }
209                 Chromosome c=population.get(i);
210                 cur = System.currentTimeMillis();
211                 c.computeFitness();
212                 progress.set(0.6*(i*1.0/popSize));
213                 totalTime += (System.currentTimeMillis() - cur) / 1000.0;
214             }
215             System.out.println("fitness avg time: " + (totalTime / population.size()));
216             cur = System.currentTimeMillis();
217 
218             if(!isRunning){
219                 progress.set(0);
220                 return;
221             }
222             //selection
223             population = selection(elitismCount);
224             if(!isRunning){
225                 progress.set(0);
226                 return;
227             }
228             System.out.println("selection: " + (System.currentTimeMillis() - cur) / 1000.0);
229             Chromosome solution = null;
230             for (int j = 0; j < popSize; j++) {
231                 Chromosome temp = population.get(j);
232 //                System.out.println(temp.getFitness());
233                 if (solution == null || solution.getFitness() < temp.getFitness()) {
234                     solution = temp;
235                 }
236                 progress.set(0.8+(0.2*j/popSize));
237             }
238 //            System.out.println("...");
239             if(!isRunning){
240                 progress.set(0);
241                 return;
242             }
243 
244             this.solutionList.add(solution);
245             System.out.println(solution.getFitness());
246 
247             if (this.solutionList.size() > convergeGen) {
248                 double delta = 0;
249                 this.solutionList.remove(0);
250                 for (int j = 1; j < this.solutionList.size(); j++) {
251                     double curr = this.solutionList.get(j).getFitness();
252                     double prev = this.solutionList.get(j - 1).getFitness();
253                     delta += Math.abs(curr - prev);
254                 }
255                 delta /= this.solutionList.size();
256                 if (delta < convergeEpsilon) {
257                     a=Params.getInstance().getMaxIt();
258                 }
259             }
260             progress.set(1);
261             if(!isRunning){
262                 progress.set(0);
263                 return;
264             }
265         }
266     }
267 
268     public Chromosome getSolution(){
269         return this.solutionList.get(this.solutionList.size()-1);
270     }
271 
272     public void reset(){
273         instance=null;
274     }
275 }
```

Listing A.6: Gene.java

```
1 package ga.clustering.gui.GA;
2 
3 
4 import ga.clustering.gui.IR.Vector;
5 import java.util.HashMap;
6 
```

```java
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */

/**
 *
 * @author CorneliusDavid
 */
public class Gene {
    private Vector value;

    public Gene(Vector value) {
        this.value = new Vector(value);
    }

    public Gene(Gene g){
        this.value=new Vector(g.value);
    }

    public Vector getValue() {
        return value;
    }

    public void mutate(){
        value.mutate();
    }
}
```

Listing A.7: CosineSimilarityCalculator.java

```java
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package ga.clustering.gui.IR;

import java.util.Iterator;
import java.util.Set;

/**
 *
 * @author CorneliusDavid
 */
public class CosineSimilarityCalculator extends SimilarityCalculator{

    @Override
    public double calculateSimilarity(Vector vector1, Vector vector2) {
//        System.out.println("l1 "+vsm1.getLength());
//        System.out.println("l2 "+vsm2.getLength());
//        System.out.println("dot "+dotProduct(vsm1, vsm2));
//        if(vsm1.getLength()==0 || vsm2.getLength()==0){
//            System.out.println("asdasad");
//            return 0;
//        }
        return dotProduct(vector1, vector2)/(vector1.getLength()*vector2.getLength());
    }

    private double dotProduct(Vector vector1, Vector vector2){
        Set<String> terms=vector1.getKeySet();
        Iterator<String> it=terms.iterator();
        double result=0;
        while(it.hasNext()){
            String term=it.next();
            double weight1=vector1.getWeight(term);
            double weight2=vector2.getWeight(term);
            result+=(weight1*weight2);
        }
        return result;
    }
}
```

Listing A.8: Document.java

```java
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package ga.clustering.gui.IR;

import java.io.ByteArrayInputStream;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.io.InputStreamReader;
import java.nio.charset.StandardCharsets;
import java.util.HashMap;
import java.util.Scanner;
import org.apache.pdfbox.pdmodel.PDDocument;
import org.apache.pdfbox.text.PDFTextStripper;

/**
```

```
 *
 * @author CorneliusDavid
 */
public class Document {
    private File file;
    protected HashMap<String,Integer> wordCount;
    private Vector vector;

    public Document(File file){
        this.file=file;
        wordCount=new HashMap<>();

        try {
            this.indexDocument();
        } catch (Exception ex) {}

        this.vector=new Vector(wordCount);//TODO: ganti param
    }

    private void indexDocument() throws FileNotFoundException, IOException {
        Scanner sc=new Scanner(new InputStreamReader(new FileInputStream(file)));
        if(file.getName().substring(file.getName().lastIndexOf(".")+1).equalsIgnoreCase("pdf")){
            PDDocument doc=PDDocument.load(file);
            PDFTextStripper pdfts=new PDFTextStripper();
            String input=pdfts.getText(doc);
            sc=new Scanner(new ByteArrayInputStream(input.getBytes(StandardCharsets.UTF_8)));
            doc.close();
        }
//        StemmerIndo stemmer=new StemmerIndo();
        while(sc.hasNext())
        {
            String temp=sc.next();
            temp=temp.replaceAll("[^A-Za-z0-9]", "").toLowerCase();
            if(temp.length()==0)continue;
//            if(ClusteringConfig.getInstance().USE_STEMMER){
//                temp=stemmer.getRootWord(temp);
//            }
//            if(ClusteringConfig.getInstance().STOPWORD_REMOVAL){
//                if(Lexicon.getInstance().isStopWord(temp)){
//                    continue;
//                }
//            }
            Lexicon.getInstance().insertTerm(temp);
            if (!wordCount.containsKey(temp)) {
                wordCount.put(temp, 0);
                Lexicon.getInstance().updateDF(temp);
            }
            wordCount.put(temp, wordCount.get(temp) + 1);
        }
    }

    public int getWordCount(String term) {
        if(!wordCount.containsKey(term)){
            return 0;
        }
        return wordCount.get(term).intValue();
    }

    public Vector getVector() {
        return vector;
    }

    public String getDocName(){
        return file.getAbsolutePath();
    }
}
```

Listing A.9: FrequencyWeighting.java

```
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package ga.clustering.gui.IR;

import java.util.HashMap;

/**
 *
 * @author Cornelius David
 */
public class FrequencyWeighting extends TermWeighting{

    @Override
    public double calculateWeight(String term, HashMap<String, Integer> wordCount) {
        int frequency=wordCount.get(term);
        return frequency*1.0;
    }

}
```

Listing A.10: Lexicon.java

```
/*
 * To change this license header, choose License Headers in Project Properties.
```

```java
 * To change this termlate file, choose Tools | Templates
 * and open the termlate in the editor.
 */
package ga.clustering.gui.IR;

import java.util.HashMap;
import java.util.Set;

/**
 *
 * @author CorneliusDavid
 */
public class Lexicon {

    private HashMap<String, Integer> globalTermCount;
    private HashMap<String, Integer> documentFrequency;
//    private HashSet<String> stopwords;
    private static Lexicon instance;
    private int numberOfDocument;

    private Lexicon() {
        this.globalTermCount = new HashMap<>();
//        this.stopwords=new HashSet<>();
        this.documentFrequency=new HashMap<>();
//        this.loadStopwords();
    }

//    private void loadStopwords(){
//        try {
//            BufferedReader br=new BufferedReader(new FileReader("stopword-list.txt"));
//            String tmp=br.readLine();
//            while(tmp!=null && tmp.length()>0){
//                this.stopwords.add(tmp);
//                tmp=br.readLine();
//            }
//        } catch (Exception ex) {}
//    }

    public static Lexicon getInstance() {
        if (instance == null) {
            instance = new Lexicon();
        }
        return instance;
    }

    /**
     * memasukkan term ke dalam
     *
     * @param term String yang akan dimasukkan
     */
    public void insertTerm(String term) {
        if (!globalTermCount.containsKey(term)) {
            globalTermCount.put(term, 0);
        }
        globalTermCount.put(term, globalTermCount.get(term) + 1);
    }

    public Set<String> getAllTermList(){
        return this.globalTermCount.keySet();
    }

    public void updateDF(String term){
        if(!documentFrequency.containsKey(term)){
            documentFrequency.put(term, 0);
        }
        documentFrequency.put(term, documentFrequency.get(term) + 1);
    }

//    public double getMaxValue(){
//        double max=0;
//        String str="";
//        for(Entry<String,Integer> x:globalTermCount.entrySet()){
//            if(max<x.getValue()){
//                max=x.getValue();
//                str=x.getKey();
//            }
//        }
//        return max;
//    }

//    public boolean isStopWord(String term){
//        return stopwords.contains(term);
//    }

    public int getNumberOfDocument() {
        return numberOfDocument;
    }

    public void setNumberOfDocument(int numberOfDocument) {
        this.numberOfDocument = numberOfDocument;
    }

    public int getDocumentFrequency(String term){
        return this.documentFrequency.get(term);
    }
}
```

Listing A.11: SimilarityCalculator.java

```java
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package ga.clustering.gui.IR;

/**
 *
 * @author CorneliusDavid
 */
public abstract class SimilarityCalculator {
    public abstract double calculateSimilarity(Vector vector1,Vector vector2);
}
```

Listing A.12: TermWeighting.java

```java
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package ga.clustering.gui.IR;

import java.util.HashMap;

/**
 *
 * @author Cornelius David
 */
public abstract class TermWeighting {

    public abstract double calculateWeight(String term, HashMap<String, Integer> wordCount);
}
```

Listing A.13: TFIDFWeighting.java

```java
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package ga.clustering.gui.IR;

import java.util.HashMap;
import java.util.Map;

/**
 *
 * @author Cornelius David
 */
public class TFIDFWeighting extends TermWeighting{

    @Override
    public double calculateWeight(String term, HashMap<String, Integer> wordCount) {
        return calculateTF(term, wordCount)*calculateIDF(term);
    }

    private double calculateTF(String term, HashMap<String, Integer> wordCount){
        int frequency=wordCount.get(term);
        int sumOfFrequency=0;
        for (Map.Entry<String, Integer> entry : wordCount.entrySet()) {
            sumOfFrequency+=entry.getValue();
        }
        return frequency*1.0/sumOfFrequency;
    }

    private double calculateIDF(String term){
        int docFreq=Lexicon.getInstance().getDocumentFrequency(term);
        int N=Lexicon.getInstance().getNumberOfDocument();

        return Math.log(N*1.0/docFreq)/Math.log(2);
    }

}
```

Listing A.14: Vector.java

```java
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package ga.clustering.gui.IR;

import ga.clustering.gui.Params;
import java.util.HashMap;
import java.util.Map;
import java.util.Random;
import java.util.Set;

/**
 *
 * @author CorneliusDavid
```

```
 */
public class Vector {

    private HashMap<String, Double> termsWeight;
    private SimilarityCalculator similarityCalculator;
    private TermWeighting termWeighting;


    public Vector(HashMap<String, Integer> wordCount) {
        termsWeight = new HashMap<>();

        similarityCalculator = new CosineSimilarityCalculator();
        if(Params.getInstance().getWeightingMethod()==0){
            termWeighting=new TFIDFWeighting();
        }else{
            termWeighting=new FrequencyWeighting();
        }

        generateVector(wordCount);
    }

    public Vector(Vector v){
        similarityCalculator = new CosineSimilarityCalculator();
        if(Params.getInstance().getWeightingMethod()==0){
            termWeighting=new TFIDFWeighting();
        }else{
            termWeighting=new FrequencyWeighting();
        }
        this.termsWeight=new HashMap<>(v.getTermsWeight());
    }

    private void generateVector(HashMap<String, Integer> wordCount){
        Set<Map.Entry<String,Integer>> entrySet=wordCount.entrySet();
        for(Map.Entry<String,Integer> entry:entrySet){
            this.termsWeight.put(entry.getKey(), calculateWeight(entry.getKey(), wordCount));
        }
    }

    public double getWeight(String term){
        if(!termsWeight.containsKey(term))return 0.0;
        return termsWeight.get(term);
    }

    public double calculateWeight(String term, HashMap<String, Integer> wordCount){
        return termWeighting.calculateWeight(term, wordCount);
    }

    public double calculateSimilarity(Vector otherVector) {
        return similarityCalculator.calculateSimilarity(this, otherVector);
    }

    public void mutate(){
        Random rand=new Random();
//        System.out.println("tw: "+termsWeight.size());
        String key=(String) termsWeight.keySet().toArray()[rand.nextInt(termsWeight.size())];
        double value=termsWeight.get(key);
        double newVal=value+(rand.nextBoolean()?value*2*rand.nextDouble():value*-2*rand.nextDouble());
//        System.out.println(key);
        termsWeight.put(key, newVal);//mutasi dari 0 - nilai sendiri dikali 2
    }

    public void setTermsWeight(HashMap<String, Double> termsWeight) {
        this.termsWeight = new HashMap<>(termsWeight);
    }

    public void setWeight(String term, double value){
        termsWeight.put(term,value);
    }

    public double getLength(){
        double res=0.0;
        for(Map.Entry<String,Double> entry: termsWeight.entrySet()){
            res+=(entry.getValue()*entry.getValue());
        }
        res=Math.sqrt(res);
        return res;
    }

    public Set<String> getKeySet(){
        return termsWeight.keySet();
    }

    public HashMap<String, Double> getTermsWeight() {
        return termsWeight;
    }

    public int getDimension(){
        return this.termsWeight.size();
    }
}
```

Listing A.15: KMeans.java

```
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
```

```java
package ga.clustering.gui.KMeans;


import ga.clustering.gui.Params;
import ga.clustering.gui.IR.Document;
import ga.clustering.gui.IR.Lexicon;
import ga.clustering.gui.IR.Vector;
import java.io.File;
import java.util.Collections;
import java.util.HashMap;
import java.util.LinkedList;
import java.util.List;
import java.util.Queue;
import javafx.beans.property.ReadOnlyDoubleProperty;
import javafx.beans.property.ReadOnlyDoubleWrapper;

/**
 *
 * @author Cornelius David
 */
public class KMeans {

    private List<Document> docs;
    private HashMap<Document, Integer> solution;
    private static KMeans instance;
    private double solutionIntracluster;

    private boolean isRunning;

    private final ReadOnlyDoubleWrapper progress = new ReadOnlyDoubleWrapper();

    private KMeans(){
        docs=new LinkedList<>();
        solutionIntracluster=-1;
        this.isRunning=true;
    }

    public void setIsRunning(boolean isRunning) {
        this.isRunning = isRunning;
    }

    public double getProgress() {
        return progressProperty().get();
    }

    public ReadOnlyDoubleProperty progressProperty() {
        return progress ;
    }

    public HashMap<Document, Integer> getSolution() {
        return solution;
    }

    public static KMeans getInstance(){
        if(instance==null)instance=new KMeans();
        return instance;
    }

    public void cluster() {
        int k=Params.getInstance().getK();
        String filepath=Params.getInstance().getFilepath();
        int maxIt=Params.getInstance().getMaxIt();
        File folder = new File(filepath);
        File[] files = folder.listFiles();
        Queue<File> queue = new LinkedList<>();
        for (int i = 0; i < files.length; i++) {
            queue.offer(files[i]);
        }
        LinkedList<File> allFiles = new LinkedList<>();

        while (!queue.isEmpty()) {
            File tmp = queue.poll();
            if (tmp.isDirectory()) {
                File[] tmpFolder = tmp.listFiles();
                for (int i = 0; i < tmpFolder.length; i++) {
                    queue.offer(tmpFolder[i]);
                }
            } else {
                allFiles.add(tmp);
            }
        }

        Lexicon.getInstance().setNumberOfDocument(allFiles.size());

        for (File file : allFiles) {
            docs.add(new Document(file));
        }

        //init cluster
        List<Document> list = new LinkedList<>();
        list.addAll(docs);
        Collections.shuffle(list);
        Vector[] centroids = new Vector[k];
        for (int i = 0; i < centroids.length; i++) {
            centroids[i] = list.get(i).getVector();
        }

        System.out.println("init_done");

```

```
105         if(!isRunning){
106             progress.set(0);
107             return;
108         }
109 
110         iteration:
111         for (int a = 0; a < maxIt; a++) {
112             System.out.println("it-"+(a+1));
113             HashMap<Document, Integer> cluster=determineCluster(docs, centroids);
114             System.out.println("assign");
115             progress.set(0);
116 
117             if(!isRunning){
118                 progress.set(0);
119                 return;
120             }
121 
122             HashMap<String, Double> sumOfCentroid[] = new HashMap[k];
123             for (int i = 0; i < k; i++) {
124                 sumOfCentroid[i]=new HashMap<>();
125             }
126             int pointCount[] = new int[k];
127             for (int i = 0; i < docs.size(); i++) {
128                 int clusterCode = cluster.get(docs.get(i));
129                 for (String term : docs.get(i).getVector().getKeySet()) {
130                     double curValue = sumOfCentroid[clusterCode].containsKey(term) ? sumOfCentroid[clusterCode].get(term) : 0;
131                     curValue += docs.get(i).getVector().getWeight(term);
132                     if (curValue != 0) {
133                         sumOfCentroid[clusterCode].put(term, curValue);
134                     }
135                 }
136                 if(!isRunning){
137                     progress.set(0);
138                     return;
139                 }
140                 pointCount[clusterCode]++;
141                 progress.set(i*0.5/docs.size());
142             }
143             System.out.println("recompute");
144             for (int i = 0; i < k; i++) {
145                 HashMap<String,Double> temp=new HashMap<>();
146                 for (String term : Lexicon.getInstance().getAllTermList()) {
147                     double nextValue = sumOfCentroid[i].containsKey(term) ? sumOfCentroid[i].get(term) : 0;
148                     if(!isRunning){
149                         progress.set(0);
150                         return;
151                     }
152                     nextValue = pointCount[i] == 0.0 ? 0.0 : nextValue / pointCount[i];
153                     if (nextValue != 0) {
154 //                        centroids[i].setWeight(term, nextValue);
155                         temp.put(term, nextValue);
156                     }
157                 }
158                 centroids[i].setTermsWeight(temp);
159                 progress.set(0.5+(i*0.49)/k);
160             }
161 //            System.out.println(Arrays.toString(pointCount));
162             System.out.println(computeIntracluster(cluster, centroids));
163             if(!isRunning){
164                 progress.set(0);
165                 return;
166             }
167             progress.set(1);
168             if(solution==null)solution=cluster;
169             else{
170                 for (int i = 0; i < docs.size(); i++) {
171                     int before=solution.get(docs.get(i));
172                     int cur=cluster.get(docs.get(i));
173                     if(before!=cur){
174                         solution=cluster;
175                         continue iteration;
176                     }
177                 }
178                 solutionIntracluster=computeIntracluster(cluster, centroids);
179                 return;
180             }
181         }
182         solutionIntracluster=computeIntracluster(solution, centroids);
183     }
184 
185     public double getSolutionIntracluster() {
186         return solutionIntracluster;
187     }
188 
189     private double computeIntracluster(HashMap<Document, Integer> cluster, Vector[] centroids){
190         double res=0.0;
191 //        determineCluster(docs, centroids);
192         for (int i = 0; i < docs.size(); i++) {
193             int clusterCode=cluster.get(docs.get(i));
194             double tmp=docs.get(i).getVector().calculateSimilarity(centroids[clusterCode]);
195             res+=tmp;
196         }
197         return res;
198     }
199 
200     private HashMap<Document, Integer> determineCluster(List<Document> docs, Vector[] centroids){
201         HashMap<Document, Integer> clusteringResult=new HashMap<>();
202         for (int i = 0; i < docs.size(); i++) {
203             int cluster=-1;
```

```
204             double similarity=Double.MIN_VALUE;
205             for (int j = 0; j < centroids.length; j++) {
206                 double temp=docs.get(i).getVector().calculateSimilarity(centroids[j]);
207                 if(temp>similarity){
208                     similarity=temp;
209                     cluster=j;
210                 }
211             }
212             clusteringResult.put(docs.get(i), cluster);
213         }
214         return clusteringResult;
215     }
216 
217     public void reset(){
218         instance=null;
219     }
220 }
```

LAMPIRAN B

HASIL EKSPERIMEN

Hasil eksperimen algoritma genetika:

1. Kasus uji 1 (parameter ideal)
   Parameter:

   - Banyaknya *Cluster*: 5
   - Banyaknya Populasi: 100
   - Metode Pembobotan: TF-IDF
   - Probabilitas Mutasi: 0.05
   - Maksimum Iterasi it: 100
   - Individu Elitisme: 1
   - Banyaknya Generasi Konvergen: 3
   - Batas Konvergen: 0.00001

   Hasil eksperimen berdasarkan parameter diatas dijelaskan dalam Tabel B.1.

Waktu	Intercluster	Iterasi	Purity
1 jam 2 menit 9 detik	781.4455224	5	0.802696629
1 jam 5 menit 3 detik	533.7897158	5	0.820224719
1 jam 1 menit 31 detik	781.7710303	4	0.794157303
1 jam 9 menit 21 detik	781.5705197	5	0.78741573
1 jam 9 menit 44 detik	779.5194513	4	0.78741573

Tabel B.1: Hasil eksperimen kasus uji 1 (parameter ideal)

2. Kasus uji 2 (Populasi=50)
   Parameter:

   - Banyaknya *Cluster*: 5
   - Banyaknya Populasi: 50
   - Metode Pembobotan: TF-IDF
   - Probabilitas Mutasi: 0.05
   - Maksimum Iterasi it: 100
   - Individu Elitisme: 1
   - Banyaknya Generasi Konvergen: 3
   - Batas Konvergen: 0.00001

Hasil eksperimen berdasarkan parameter diatas dijelaskan dalam Tabel B.2.

Waktu	Intercluster	Iterasi	Purity
34 menit 49 detik	537.8835842	4	0.770786517
50 menit 13 detik	922.3475657	6	0.708764045
54 menit 6 detik	782.7303944	6	0.806292135
37 menit 26 detik	779.3514206	4	0.785617978
35 menit 36 detik	780.2768031	4	0.826516854

Tabel B.2: Hasil eksperimen kasus uji 2 (Populasi=50)

3. Kasus uji 3 (Populasi=150)
   Parameter:

   - Banyaknya *Cluster*: 5
   - Banyaknya Populasi: 150
   - Metode Pembobotan: TF-IDF
   - Probabilitas Mutasi: 0.05
   - Maksimum Iterasi it: 100
   - Individu Elitisme: 1
   - Banyaknya Generasi Konvergen: 3
   - Batas Konvergen: 0.00001

   Hasil eksperimen berdasarkan parameter diatas dijelaskan dalam Tabel B.3.

Waktu	Intercluster	Iterasi	Purity
1 jam 40 menit 56 detik	781.0071837	4	0.758651685
1 jam 38 menit 5 detik	545.8893228	4	0.96
1 jam 22 menit 4 detik	920.7411105	4	0.623820225
2 jam 10 menit 59 detik	542.7638178	5	0.965393258
2 jam 25 menit 13 detik	1520.496586	6	0.425617978

Tabel B.3: Hasil eksperimen kasus uji 3 (Populasi=150)

4. Kasus uji 4 (Bobot frekuensi)
   Parameter:

   - Banyaknya *Cluster*: 5
   - Banyaknya Populasi: 100
   - Metode Pembobotan: Frekuensi
   - Probabilitas Mutasi: 0.05
   - Maksimum Iterasi it: 100
   - Individu Elitisme: 1
   - Banyaknya Generasi Konvergen: 3
   - Batas Konvergen: 0.00001

   Hasil eksperimen berdasarkan parameter diatas dijelaskan dalam Tabel B.4.

Waktu	Intercluster	Iterasi	Purity
2 jam 44 menit 10 detik	1750.439783	5	0.581123596
2 jam 42 menit 52 detik	1750.902724	8	0.57258427
2 jam 13 menit 36 detik	1750.920796	6	0.543370787
2 jam 34 menit 57 detik	1751.049984	7	0.583820225
3 jam 24 menit 23 detik	1751.061378	9	0.575730337

Tabel B.4: Hasil eksperimen kasus uji 4 (Bobot frekuensi)

5. Kasus uji 5 (Probabilitas mutasi=0)
   Parameter:

   - Banyaknya *Cluster*: 5
   - Banyaknya Populasi: 100
   - Metode Pembobotan: TF-IDF
   - Probabilitas Mutasi: 0
   - Maksimum Iterasi it: 100
   - Individu Elitisme: 1
   - Banyaknya Generasi Konvergen: 3
   - Batas Konvergen: 0.00001

   Hasil eksperimen berdasarkan parameter diatas dijelaskan dalam Tabel B.5.

Waktu	Intercluster	Iterasi	Purity
2 jam 11 menit 4 detik	779.4871027	6	0.893932584
1 jam 14 menit 37 detik	543.1994982	4	0.955955056
1 jam 54 menit 46 detik	1610.403312	6	0.361348315
1 jam 1 menit 17 detik	920.2416313	4	0.595955056
1 jam 53 menit 29 detik	1675.987642	6	0.354606742

Tabel B.5: Hasil eksperimen kasus uji 5 (Probabilitas mutasi=0)

6. Kasus uji 6 Probabilitas mutasi=0.25)
   Parameter:

   - Banyaknya *Cluster*: 5
   - Banyaknya Populasi: 100
   - Metode Pembobotan: TF-IDF
   - Probabilitas Mutasi: 0.25
   - Maksimum Iterasi it: 100
   - Individu Elitisme: 1
   - Banyaknya Generasi Konvergen: 3
   - Batas Konvergen: 0.00001

   Hasil eksperimen berdasarkan parameter diatas dijelaskan dalam Tabel B.6.

Waktu	Intercluster	Iterasi	Purity
1 jam 16 menit 39 detik	1206.317545	5	0.45752809
1 jam 32 menit 52 detik	1393.180775	6	0.414382022
1 jam 48 menit 20 detik	921.4314001	5	0.567191011
1 jam 23 menit 52 detik	1204.926873	4	0.344719101
1 jam 28 menit 9 detik	1726.000931	5	0.401797753

Tabel B.6: Hasil eksperimen kasus uji 6 (Probabilitas mutasi=0.25)

7. Kasus uji 7 (Individu eltisme=0)
   Parameter:
   - Banyaknya *Cluster*: 5
   - Banyaknya Populasi: 100
   - Metode Pembobotan: TF-IDF
   - Probabilitas Mutasi: 0.05
   - Maksimum Iterasi it: 100
   - Individu Elitisme: 0
   - Banyaknya Generasi Konvergen: 3
   - Batas Konvergen: 0.00001
8. Kasus uji 8 (Individu elitisme=5)
   Parameter:
   - Banyaknya *Cluster*: 5
   - Banyaknya Populasi: 100
   - Metode Pembobotan: TF-IDF
   - Probabilitas Mutasi: 0.05
   - Maksimum Iterasi it: 100
   - Individu Elitisme: 5
   - Banyaknya Generasi Konvergen: 3
   - Batas Konvergen: 0.00001

Hasil eksperimen algoritma K-means:
Parameter:

- Banyaknya *Cluster*: 5
- Metode Pembobotan: TF-IDF
- Maksimum Iterasi it: 100

Hasil eksperimen berdasarkan parameter diatas dijelaskan dalam Tabel B.7.

Waktu	Intercluster	Iterasi	Purity
1 menit 30 detik	779.1467826	12	0.734382022
1 menit 2 detik	512.6894755	7	0.456179775
45 detik	903.4065026	7	0.252080274
1 menit 5 detik	1208.220743	10	0.348314607
2 menit 19 detik	1383.188573	22	0.276853933
1 menit 32 detik	776.9612952	12	0.625168539
2 menit 12 detik	543.139087	14	0.746966292
1 menit 31 detik	785.1360162	10	0.794606742
1 menit 47 detik	523.7897906	13	0.636404494
58 detik	900.2035951	8	0.248153619

Tabel B.7: Hasil eksperimen algoritma K-means