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

LEMBAR PENGESAHAN

PENGELOMPOKAN DOKUMEN BERBASIS ALGORITMA GENETIKA

Cornelius David Herianto

NPM: 2015730034

Bandung, 22 Mei 2019

Menyetujui,

Pembimbing

Kristopher David Harjono, M.T.

Ketua Tim Penguji

Anggota Tim Penguji

Husnul Hakim, M.T.

Rosa De Lima, M.Kom.

Mengetahui,

Ketua Program Studi

Mariskha Tri Adithia, P.D.Eng

PERNYATAAN

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

PENGELOMPOKAN DOKUMEN BERBASIS ALGORITMA GENETIKA

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

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

Dinyatakan di Bandung,
Tanggal 22 Mei 2019

Meterai
Rp. 6000

Cornelius David Herianto
NPM: 2015730034

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 [1]. *Document clustering* (pengelompokan dokumen) merupakan proses pengelompokan yang dilakukan terhadap suatu koleksi dokumen. Pengelompokan dokumen diterapkan dalam beberapa bidang seperti penambangan web, mesin pencari (*search engine*), dan temu kembali informasi (*information retrieval*) [3]. Hal yang dilakukan dalam pengelompokan dokumen adalah mengukur kemiripan (*similarity*) antar dokumen dan mengelompokan dokumen yang serupa. 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. GA dapat digunakan untuk mengelompokkan dokumen dengan beberapa adaptasi terhadap representasi kromosom, fungsi *fitness*, seleksi, persilangan, dan mutasi.

Algoritma genetika dan algoritma *K-means* diuji menggunakan suatu *dataset* berlabel untuk membandingkan waktu dan hasil pengelompokan dari kedua algoritma tersebut. Berdasarkan hasil eksperimen menggunakan *dataset* dalam penelitian ini, rata-rata nilai *purity* dari hasil pengelompokan menggunakan algoritma genetika adalah sebesar 0.799, lebih baik 56% dibandingkan dengan menggunakan algoritma K-means. Hal ini membuktikan bahwa algoritma genetika sudah dapat mengelompokan dokumen dengan hasil yang memuaskan. 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*.

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 [1]. 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 modeled 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 datasets to compare the running time and clustering result. Based on the experimental results using the datasets in this study, the average purity value of clustering results using genetic algorithms is 0.799, 56% greater than using the K-means algorithm. This proves that the genetic algorithm is able to cluster documents with satisfactory results. But in terms of running time, genetic algorithms take 4365% more time than the K-means algorithm. This is caused by the computational process carried out on the genetic algorithm is far more complex than the K-means algorithm.

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

Dipersembahkan kepada Tuhan YME, keluarga tercinta,
dan diri sendiri

KATA PENGANTAR

Puji syukur kepada Tuhan Yang Maha Esa atas berkat yang diberikan kepada penulis sehingga dapat menyelesaikan skripsi dengan judul **Pengelompokan Dokumen Berbasis Algoritma Genetika** dengan baik dan tepat waktu. Selama menjalani proses perkuliahan dan penyusunan skripsi, penulis telah mendapat banyak bantuan dan dukungan dalam menghadapi hambatan yang ada. Oleh karena itu, penulis inign menyampaikan rasa terima kasih kepada:

1. Keluarga penulis yaitu Papa dan Mama yang selalu mendukung penulis secara moral dan materiil sehingga penulis dapat menyelesaikan proses perkuliahan dan skripsi ini dengan baik. Serta kepada Michelle dan Vincent sebagai adik penulis yang selalu mendukung penulis dalam menyelesaikan skripsi ini.

2. Bapak Kristopher David Harjono, M.T. sebagai dosen pembimbing yang telah membimbing penulis hingga dapat menyelesaikan skripsi ini.

3. Bapak Husnul Hakim, M.T. dan Ibu Rosa De Lima, M.Kom. sebagai dosen penguji yang telah membantu dalam menguji dan memperbaiki skripsi ini.

4. Ibu Mariskha Tri Adithia, P.D.Eng selaku Ketua Program Studi Teknik Informatika Fakultas Teknologi Informasi dan Sains Universitas Katolik Parahyangan.

5. Arlin Sasqia Puspa Shiffa sebagai sahabat yang selalu mendampingi penulis dalam penyusunan skripsi dari awal hingga akhir bahkan membantu penulis mempersiapkan presentasi sidang skripsi.

6. Vania Stephanie dan Khezia Josephine sebagai sahabat penulis yang senantiasa memberikan dukungan, semangat, masukkan yang berguna untuk penulis, serta menghibur penulis selama proses penyusunan skripsi ini terutama saat sedang mengalami kesulitan.

7. Teman-teman dari Grup MANAYGBILANGWGAGUNA yaitu AK, Otung, Devie, Gilbert, Hoshea, Jeane, Khen, Mark, Matthew, Oyeng, Bebe, Rifo, Rizky, Sean, Vinny, dan WM yang telah menghibur dan mendukung penulis dalam penyusunan skripsi ini.

8. Teman-teman dari Grup Korea yaitu Ario, Dandy, Fakhry, Felis, Hima, Hizkia, Irvan, Acong, Joshua, Kezia, Edrick, Ocin, Momon, Thoby, Victor, Yona, Yudhis, dan Matthew Ariel sebagai teman seperjuangan di Teknik Informatika UNPAR angkatan 2015 yang telah memberi semangat dan dukungan kepada penulis.

9. Teman-teman penulis lain yang tidak dapat disebutkan satu persatu. Terima kasih untuk segala dukungannya sehingga skripsi ini dapat diselesaikan dengan baik.

Akhir kata, semoga skripsi ini dapat bermanfaat bagi pembaca dan dapat menjadi dasar untuk penelitian yang terkait dengan skripsi ini.

Bandung, Mei 2019

Penulis

DAFTAR ISI

Kata Pengantar xv

Daftar Isi xvii

Daftar Gambar xix

Daftar Tabel xxi

1 Pendahuluan 1
- 1.1 Latar Belakang . . . 1
- 1.2 Rumusan Masalah . . . 2
- 1.3 Tujuan Penelitian . . . 2
- 1.4 Batasan Masalah . . . 2
- 1.5 Metodologi . . . 2
- 1.6 Sistematika Pembahasan . . . 3

2 Landasan Teori 5
- 2.1 Pengelompokan . . . 5
  - 2.1.1 Definisi Pengelompokan . . . 5
  - 2.1.2 Aplikasi Pengelompokan . . . 6
  - 2.1.3 *Local Optimum* . . . 7
- 2.2 *K-Means* . . . 7
- 2.3 Algoritma Genetika . . . 8
  - 2.3.1 *Fitness* . . . 9
  - 2.3.2 Seleksi . . . 10
  - 2.3.3 Persilangan . . . 11
  - 2.3.4 Mutasi . . . 11
  - 2.3.5 Proses Pencarian Dalam Algoritma Genetika . . . 11
  - 2.3.6 GA dalam Pengelompokan . . . 14
- 2.4 Model Ruang Vektor . . . 14
- 2.5 Pembobotan *Term* (*Term Weighting*) . . . 14
  - 2.5.1 Bobot frekuensi . . . 15
  - 2.5.2 Bobot TF-IDF . . . 15
- 2.6 Metrik *Intracluster* untuk Mengukur Kinerja Metode *Clustering* . . . 16
- 2.7 *Clustering Purity* . . . 17

3 Analisis 19
- 3.1 Analisis *Dataset* . . . 19
- 3.2 Representasi Dokumen . . . 19
- 3.3 Model Ruang Vektor . . . 20
  - 3.3.1 Bobot Frekuensi . . . 20
  - 3.3.2 Bobot TF-IDF . . . 20
- 3.4 Representasi Kromosom . . . 22

3.5 Fungsi *Fitness* . . . 23
3.6 Operasi Genetik Dalam Pengelompokan Dokumen . . . 24
3.6.1 Inisialisasi Populasi . . . 24
3.6.2 Seleksi . . . 24
3.6.3 Persilangan . . . 24
3.6.4 Mutasi . . . 25
3.7 Evaluasi Hasil Pengelompokan Menggunakan *Purity* . . . 26

4 Perancangan **27**
4.1 Kebutuhan Masukan dan Keluaran . . . 27
4.2 Rancangan Kelas . . . 29
4.2.1 *Document* . . . 30
4.2.2 *Vector* . . . 31
4.2.3 *SimilarityCalculator* . . . 32
4.2.4 *CosineSimilarityCalculator* . . . 32
4.2.5 *TermWeighting* . . . 33
4.2.6 *FrequencyWeighting* . . . 33
4.2.7 *TFIDFWeighting* . . . 33
4.2.8 *Lexicon* . . . 34
4.2.9 *Gene* . . . 35
4.2.10 *Chromosome* . . . 35
4.2.11 *GAClusterer* . . . 37
4.2.12 *Params* . . . 39
4.2.13 *KMeans* . . . 41
4.2.14 *FXMLDocumentController* . . . 43
4.3 Perancangan Antarmuka Pengguna . . . 45
4.3.1 Halaman Algoritma Genetika . . . 46
4.3.2 Halaman *K-Means* . . . 48

5 Pengujian dan Eksperimen **51**
5.1 Skenario Pengujian Eksperimental . . . 51
5.2 Eksperimen Algoritma Genetika . . . 53
5.3 Eksperimen *K-Means* . . . 62
5.4 Analisis Hasil Eksperimen . . . 64

6 Kesimpulan dan Saran **67**
6.1 Kesimpulan . . . 67
6.2 Saran . . . 68

Daftar Referensi **69**

A Kode Program **71**

B Hasil Eksperimen **91**

C Contoh Dataset **97**
C.1 Topik *Business* . . . 97

DAFTAR GAMBAR

2.1 Contoh *cluster* hasil pengelompokan . . . 5
2.2 Hasil pencarian (*clustered search result*) di Yippy . . . 6
2.3 *Local Optimum* dan *Global Optimum* . . . 7
2.4 Alur algoritma genetika dasar . . . 9
2.5 Ilustrasi *roulette-wheel* . . . 10
2.6 *Single-point crossover* . . . 11
2.7 *Ilustrasi untuk jarak intracluster* . . . 17

3.1 Formula representasi kromosom . . . 22
3.2 Contoh representasi *centroid* ke dalam kromosom . . . 23
3.3 Ilustrasi kromosom untuk persilangan . . . 24
3.4 Ilustrasi persilangan . . . 25
3.5 Ilustrasi mutasi kromosom . . . 25

4.1 Diagram kelas . . . 29
4.2 Kelas *Document* . . . 30
4.3 Kelas *Vector* . . . 31
4.4 Kelas *SimilarityCalculator* . . . 32
4.5 Kelas *CosineSimilarityCalculator* . . . 32
4.6 Kelas *TermWeighting* . . . 33
4.7 Kelas *FrequencyWeighting* . . . 33
4.8 Kelas *TFIDFWeighting* . . . 33
4.9 Kelas *Lexicon* . . . 34
4.10 Kelas *Gene* . . . 35
4.11 Kelas *Chromosome* . . . 35
4.12 Kelas *GAClusterer* . . . 37
4.13 Kelas *Params* . . . 39
4.14 Kelas *KMeans* . . . 41
4.15 Kelas *FXMLDocumentController* . . . 43
4.16 Rancangan antarmuka halaman algoritma genetika . . . 46
4.17 Rancangan antarmuka halaman algoritma genetika . . . 48

5.1 Grafik hubungan banyaknya populasi dengan waktu pengelompokan . . . 53
5.2 Diagram hubungan banyaknya populasi dengan *intracluster similarity* . . . 54
5.3 Diagram hubungan banyaknya populasi dengan banyaknya iterasi . . . 54
5.4 Diagram hubungan banyaknya populasi dengan nilai *purity* . . . 55
5.5 Diagram hubungan metode pembobotan dengan waktu pengelompokan . . . 55
5.6 Diagram hubungan metode pembobotan dengan *intracluster similarity* . . . 56
5.7 Diagram hubungan metode pembobotan dengan banyaknya iterasi . . . 56
5.8 Diagram hubungan metode pembobotan dengan nilai *purity* . . . 57
5.9 Diagram hubungan probabilitas mutasi dengan waktu pengelompokan . . . 57
5.10 Diagram hubungan probabilitas mutasi dengan *intracluster similarity* . . . 58
5.11 Diagram hubungan probabilitas mutasi dengan banyaknya iterasi . . . 58

5.12 Diagram hubungan probabilitas mutasi dengan nilai *purity* 59
5.13 Diagram hubungan individu elitisme dengan waktu pengelompokan 60
5.14 Diagram hubungan individu elitisme dengan *intracluster similarity* 60
5.15 Diagram hubungan individu elitisme dengan banyaknya iterasi 61
5.16 Diagram hubungan individu elitisme dengan nilai *purity* 61
5.17 Diagram hubungan algoritma dengan waktu tempuh 62
5.18 Diagram hubungan algoritma dengan *intracluster similarity* 63
5.19 Diagram hubungan algoritma dengan banyaknya iterasi 63
5.20 Diagram hubungan algoritma dengan nilai *purity* 64

DAFTAR TABEL

2.1 *Term-document incidence matrix* 15
2.2 Tabel hasil pengelompokan 18

3.1 Hasil perhitungan bobot frekuensi 20
3.2 Hasil perhitungan TF 21
3.3 Hasil perhitungan IDF 21
3.4 Hasil perhitungan bobot TF-IDF 22

4.1 Contoh keluaran dalam bentuk *file CSV* 28
4.2 Rincian *field* pada halaman algoritma genetika 47
4.3 Rincian *field* pada halaman algoritma *K-means* 49

5.1 Variasi nilai variabel bebas 52
5.2 Rata-rata hasil pengelompokan dengan variasi variabel banyaknya populasi 53
5.3 Rata-rata hasil pengelompokan dengan variasi variabel metode pembobotan 55
5.4 Rata-rata hasil pengelompokan dengan variasi variabel probabilitas mutasi 57
5.5 Rata-rata hasil pengelompokan dengan variasi variabel individu elitisme 60
5.6 Rata-rata hasil pengelompokan dengan menggunakan algoritma *K-means* 62
5.7 Hasil perhitungan statistika terhadap hasil eksperimen algoritma genetika dan *K-means* 64

B.1 Hasil eksperimen kasus uji 1 (parameter ideal) 91
B.2 Hasil eksperimen kasus uji 2 (Populasi=50) 92
B.3 Hasil eksperimen kasus uji 3 (Populasi=150) 92
B.4 Hasil eksperimen kasus uji 4 (Bobot frekuensi) 93
B.5 Hasil eksperimen kasus uji 5 (Probabilitas mutasi=0) 93
B.6 Hasil eksperimen kasus uji 6 (Probabilitas mutasi=0.25) 94
B.7 Hasil eksperimen kasus uji 7 (Individu elitisme=0) 94
B.8 Hasil eksperimen kasus uji 8 (Individu elitisme=5) 95
B.9 Hasil eksperimen algoritma K-means 95

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 [2].

Document clustering (pengelompokan dokumen) merupakan proses pengelompokan yang dilakukan terhadap suatu koleksi dokumen. Pengelompokan dokumen diterapkan dalam beberapa bidang seperti penambangan web, mesin pencari (*search engine*), dan temu kembali informasi (*information retrieval*) [3]. Hal yang dilakukan dalam pengelompokan dokumen adalah mengukur kemiripan (*similarity*) antar dokumen dan mengelompokan dokumen yang serupa. Suatu dokumen dapat terdiri dari beberapa jenis informasi seperti teks, jenis tulisan, ukuran tulisan, warna tulisan, dan gambar.

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 [4].

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 [5] berdasarkan prinsip *survival of the fittest*. Algoritma ini dinamakan demikian karena menggunakan konsep-konsep dalam genetika sebagai model pemecahan masalahnya [6].

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 [6]. 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 difokuskan dengan memperlihatkan batasan masalah sebagai berikut:

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

1.5 Metodologi

Langkah-langkah yang 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 dijadikan *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 melakukan implementasi dan pengujian kembali sampai didapatkan hasil yang sudah sesuai dengan harapan.

1.6 Sistematika Pembahasan

Dokumentasi dari penelitian ini 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

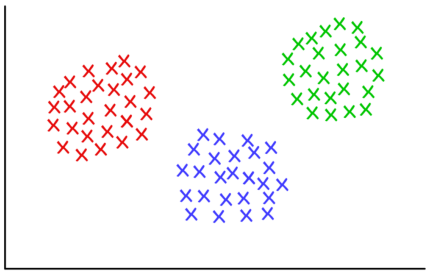
LANDASAN TEORI

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

2.1 Pengelompokan

2.1.1 Definisi Pengelompokan

Pengelompokan (*clustering*) merupakan sebuah metode untuk menggabungkan himpunan objek ke dalam kelompok-kelompok sedemikian rupa sehingga objek dalam satu kelompok (*cluster*) lebih mirip (karena suatu hal) satu sama lain daripada objek di kelompok lain [1]. 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 dibentuk. Keluaran yang dihasilkan dari proses pengelompokan adalah kelompok objek yang telah dibentuk beserta anggotanya. Setiap objek dikelompokkan berdasarkan kesamaan tertentu. Sebagai ilustrasi dari pengelompokan (Gambar 2.1), terdapat tiga *cluster* yang ditandai dengan warna merah, biru, dan hijau. Objek-objek yang berwarna sama dianggap mirip sehingga dimasukkan ke dalam kelompok yang sama. Begitu juga dengan objek yang berbeda warna dianggap tidak mirip sehingga perlu dipisahkan.



Gambar 2.1: Contoh *cluster* hasil pengelompokan

2.1.2 Aplikasi Pengelompokan

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

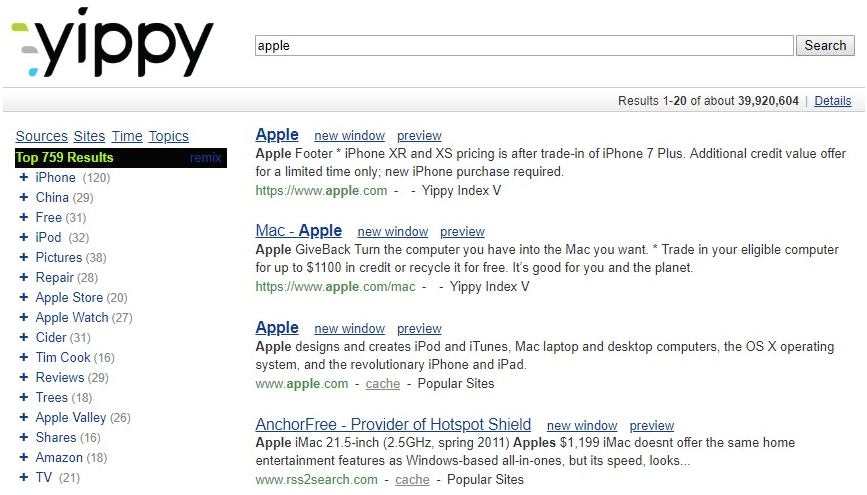
Recommender System

Recommender system adalah suatu sistem yang berfungsi untuk memprediksikan keinginan pengguna berdasarkan masukkan yang diberikan oleh pengguna [7]. Ada dua jenis sistem rekomendasi yaitu *Content-based Recommendation* dan *Collaborative Filtering*.

- *Content-based Recommendation*: mempelajari apa yang pengguna sukai lalu mencari objek lain yang mungkin juga 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*)[8]. 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].



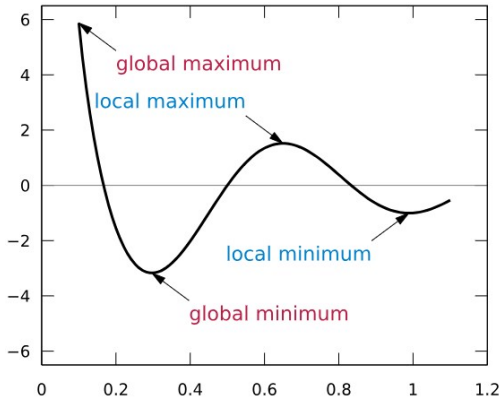
Gambar 2.2: Hasil pencarian (*clustered search result*) di Yippy

[1] https://yippy.com/

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

2.1.3 *Local Optimum*

Local optimum adalah suatu solusi yang optimal (baik maksimal maupun minimal) di antara 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 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 objek pada *cluster* ke-i, dan x_q merupakan titik ke-q pada *cluster* ke-i.

Algoritma 1 *K-MEANS*

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

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

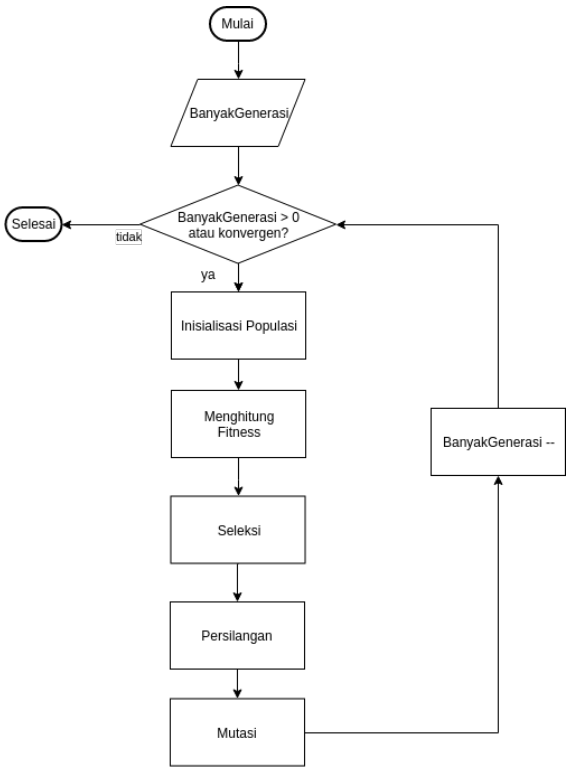
Penjelasan dari Algoritma 1 adalah sebagai berikut:

- Pada baris 1, *centroid* awal dipilih secara acak dari objek-objek yang sudah ada.
- Pada baris 4, *centroid* baru ditentukan dengan menggunakan Persamaan 2.1.
- Pada baris 5, pengulangan 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, dan lain lain [6]. Bahkan setiap individu dalam spesies yang sama pun harus bersaing menarik lawan jenis untuk berkembang biak. Individu yang kurang baik memiliki peluang bertahan hidup lebih kecil, dan individu yang bisa beradaptasi dengan baik atau "*fit*" menghasilkan keturunan dengan jumlah yang 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 digunakan dalam operasi genetik selanjutnya.

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

4. **Persilangan**
   Individu yang terpilih pada proses seleksi disilangkan untuk menghasilkan keturunan. Proses persilangan ini terjadi antara dua individu 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 diulang terus-menerus sampai ditemukan suatu solusi yang optimal atau sudah mencapai batas maksimum generasi. Berikut merupakan penjelasan lebih lanjut mengenai istilah yang ada pada GA.

2.3.1 *Fitness*

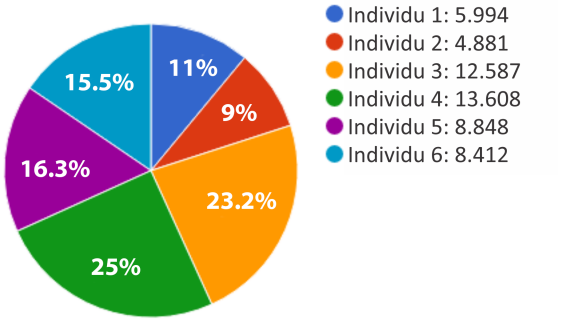
Fitness dari suatu individu dalam algoritma genetika adalah suatu nilai fungsi objektif untuk fenotipenya [6]. *Fitness* harus bisa memperkirakan seberapa dekat sebuah calon solusi dengan

solusi yang optimal. Suatu solusi yang optimal memaksimalkan fungsi *fitness*. Perhitungan *fitness* sangatlah tergantung dengan masalah yang ingin diselesaikan. Fungsi *fitness* yang baik harus bisa menentukan solusi mana yang paling baik di antara banyak calon solusi.

2.3.2 Seleksi

Seleksi adalah proses pemilihan dua induk dari populasi untuk disilangkan [6]. Tujuan dari proses seleksi adalah untuk menonjolkan individu yang memiliki nilai *fitness* tinggi dalam populasi dengan harapan keturunannya 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 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 secara acak 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 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 dengan cara normalisasi.
- Kemudian sebuah kromosom dipilih secara acak dengan cara yang sama seperti prinsip kerja roda *roulette*.

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

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. Sebagai contoh dari Gambar

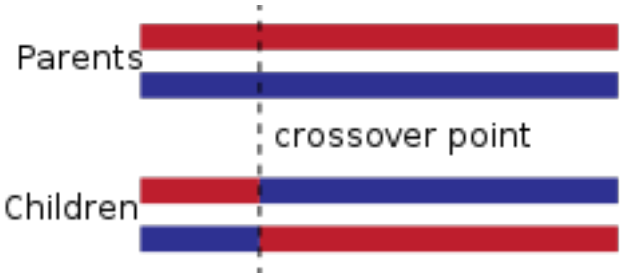
2.5, peluang individu 1 dihitung dalam Persamaan 2.3. Cara yang sama juga dapat digunakan untuk menghitung peluang dari individu 2, individu 3, dan seterusnya.

$$P_i = \frac{5.994}{5.994 + 4.881 + 12.587 + 13.608 + 8.848 + 8.412} = 0.1103 = 11\% \tag{2.3}$$

Proses seleksi dalam GA 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 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 [6]. 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* [6]. 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 dimutasi.

Jika tidak terjadi mutasi, maka keturunannya 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 dilakukan dengan mengubah nilai dari suatu gen yang telah dipilih menjadi nilai lainnya dengan teknik tertentu (tergantung struktur data dari gen yang 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 dicatat ke variabel *solusi*.
- Pada baris 9, variabel *rand* berisi bilangan riil antara 0 sampai dengan 1. Bilangan riil ini dibangkitkan secara acak dengan distribusi *uniform*.
- Pada baris 16, proses pengulangan berhenti saat nilai *fitness* pada N solusi terakhir memiliki selisih yang sangat kecil.
- 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 1, variabel n berisi banyaknya gen pada kromosom x.
- Pada baris 2, variabel c berisi bilangan bulat antara 1 sampai N. Bilangan bulat ini 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 bawah tertentu (inklusif) sampai dengan batas atas tertentu (inklusif).

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* mengembalikan nilai *fitness* dari individu yang menjadi parameternya.
- Pada baris 5, variabel *terpilih* 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 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* mengembalikan panjang dari *string* parameterya.
- Pada baris 2, fungsi *RANDOM* 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 [10]. 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 [11]. Model ruang vektor ini biasanya digunakan dalam sejumlah operasi pencarian informasi mulai dari penilaian dokumen pada *query*, klasifikasi dokumen, dan pengelompokan dokumen. Pada penerapannya, dilakukan pengukuran kemiripan suatu dokumen terhadap *query* untuk dapat menentukan peringkat relevansi dokumen terhadap *query* (*relevance ranking*). Dokumen dan *query* direpresentasikan sebagai model ruang vektor seperti pada Persamaan 2.4.

$$d_i = (w_{1,i}, w_{2,i}, ..., w_{n,i}) \tag{2.4}$$

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* 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 2.5.

$$s_{ij} = \frac{i \cdot j}{\| i \| \times \| j \|} \tag{2.5}$$

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 [11], tokenisasi merupakan proses pemotongan suatu dokumen menjadi potongan-potongan (*token*) tertentu. Pada proses yang sama, karakter tertentu 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 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 semakin kecil.

Ada beberapa cara untuk menghitung bobot suatu *term* . Dua metode yang umum digunakan di antaranya 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 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 2.6

$$w_i = tf_i \tag{2.6}$$

dengan w_i merupakan bobot *term* ke-i dan tf_i merupakan frekuensi kemunculan *term* ke-i pada dokumen. Sebagai ilustrasi, digunakan empat buah dokumen (masing-masing terdiri dari satu kalimat) yang berasal dari contoh pada [11] 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.

Tabel 2.1: *Term-document incidence matrix*

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

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 dari pembobotan frekuensi. TF-IDF merupakan gabungan dari *term frequency* (tf) dengan *inverse document frequency* (idf) untuk menghasilkan suatu bobot komposit untuk setiap *term* dalam setiap dokumen [11].

Sama seperti yang telah dijelaskan pada Subbab 2.5.1, TF merupakan banyaknya kemunculan *term* pada suatu dokumen. Namun pada prakteknya, TF pada TF-IDF merupakan pengembangan dari TF yang digunakan pada Subbab 2.5.1 yaitu TF yang dinormalisasi. TF yang telah dinormalisasi dapat dihitung menggunakan persamaan 2.7.

$$\mathrm{tf}_{t,d} = \frac{\mathrm{f}_{t,d}}{\sum_{t' \in d} \mathrm{f}_{t',d}} \tag{2.7}$$

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 memiliki frekuensi yang sangat besar pada banyak dokumen. Seperti yang sudah dijelaskan pada Subbab 2.5.1, suatu *term* yang sering muncul 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 2.8.

$$\mathrm{idf}_t = \log_2 \frac{N}{\mathrm{df}_t} \tag{2.8}$$

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 [12]. Pembobotan menggunakan TF-IDF dapat dihitung menggunakan Persamaan 2.9.

$$\text{tf-idf}_{t,d} = \mathrm{tf}_{t,d} \times \mathrm{idf}_t \tag{2.9}$$

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*

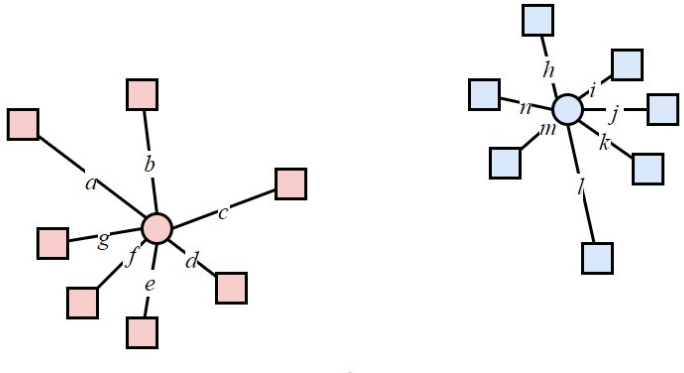
Untuk mengukur performa dari suatu algoritma, perlu dilakukan pengujian. Pengujian ini dapat menentukan apakah algoritma yang digunakan sudah cukup baik dalam menyelesaikan masalah yang dibuat. Dalam pengelompokan, ada beberapa cara untuk mengukur apakah objek-objek sudah berhasil dikelompokan secara 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 [4] 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 2.10.

$$\begin{aligned} M &= \sum_{i=1}^{K} M_i, \\ M_i &= \sum_{x_j \in C_i} \| x_j - z_i \| \end{aligned} \tag{2.10}$$

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 digunakan sebagai ilustrasi dari jarak *intracluster*.



Gambar 2.7: *Ilustrasi untuk jarak intracluster*

Pada Gambar 2.7, terdapat 14 objek yang telah dibagi ke dalam dua *cluster* berwarna merah dan biru. Persegi pada Gambar 2.7 mengilustrasikan objek yang 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 sampai dengan n. Persegi berwarna merah merupakan anggota dari *cluster* 1, sedangkan persegi berwarna biru merupakan anggota dari *cluster* 2. Berdasarkan Persamaan 2.10, 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 *Clustering 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 dijelaskan salah satu metode eksternal yaitu *purity*. *Purity* adalah persentase objek yang berhasil dikelompokan dengan benar [11]. Untuk menghitung *purity*, diperlukan suatu *dataset* yang sudah diberi label. Label ini kemudian digunakan untuk menghitung akurasi dari pengelompokan itu sendiri. Cara

untuk menghitung *purity* ditunjukkan dalam Persamaan 2.11.

$$purity = \frac{1}{N}\sum_{i=1}^{k} max_j|c_i \cap t_j| \tag{2.11}$$

dengan N merupakan banyaknya objek, k merupakan banyaknya *cluster*, c_i merupakan *cluster* ke-i, dan t_j adalah label j. Kemudian dicari jumlah terbanyak anggota c_i yang juga merupakan anggota t_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).

Sebagai contoh akan digunakan hasil pengelompokan pada Tabel 2.2. Pada Tabel 2.2, terdapat tiga buah *cluster* $c1$, $c2$, dan $c3$. Setiap *cluster* dapat berisi objek dari tiga buah label a, b, dan c. *Cluster* $c1$ berisi 3 objek berlabel a, 50 objek berlabel b, dan 10 objek berlabel c. *Cluster* $c2$ berisi 0 objek berlabel a, 1 objek berlabel b, dan 60 objek berlabel c. *Cluster* $c3$ berisi 1 objek berlabel a, 15 objek berlabel b, dan 0 objek berlabel c.

Tabel 2.2: Tabel hasil pengelompokan

	a	b	c
$c1$	3	50	10
$c2$	0	1	60
$c3$	1	15	0

Berdasarkan definisi, maka cara menghitung *purity* dari data pada Tabel 2.2 adalah dengan mencari nilai maksimal antara label a, b, dan c untuk setiap *cluster*. Setiap nilai maksimal tersebut lalu akan dijumlahkan dan dibagi dengan banyaknya objek dalam seluruh *cluster*. Pada *cluster* $c1$, nilai maksimalnya adalah 50 untuk label b. Pada *cluster* $c2$, nilai maksimalnya adalah 60 untuk label c. Sedangkan pada *cluster* $c3$, nilai maksimalnya adalah 15 untuk label b. Berdasarkan ketiga nilai tersebut, maka *purity* dari hasil pengelompokan pada Tabel 2.2 ditunjukkan dalam Persamaan 2.12.

$$purity = \frac{50 + 60 + 15}{140} = 0.892 \tag{2.12}$$

Berdasarkan hasil perhitungan yang didapatkan dari Persamaan 2.12, hasil pengelompokan pada Tabel 2.2 memiliki akurasi sebesar 89.2%. Hal ini berarti bahwa hasil pengelompokan sudah cukup baik karena hanya ada sekitar 10.8% kesalahan pengelompokan yang terjadi.

BAB 3

ANALISIS

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

3.1 Analisis *Dataset*

Pada bagian ini dibahas mengenai *dataset* yang digunakan dalam proses pengujian. *Dataset* yang 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 textitbusiness, 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.
- Tanda baca dalam dokumen diabaikan (hanya mengambil informasi berupa karakter alfanumerik)
- Kapitalisasi dalam dokumen diabaikan

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 maka 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.

Tabel 3.1: Hasil perhitungan bobot frekuensi

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

Semakin besar nilai 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. *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 2.9 pada Subbab 2.5.2. Sama dengan perhitungan bobot frekuensi, contoh yang berasal dari Subbab 3.3 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 2.7 pada Subbab 2.5.2, maka perhitungan TF dari *term* "properti" ditunjukkan pada Persamaan 3.1.

$$\text{tf}_{\text{"}properti\text{"},1} = \frac{1}{8} = 0.125 \tag{3.1}$$

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

Tabel 3.2: Hasil perhitungan TF

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

Untuk perhitungan IDF digunakan Persamaan 2.8 pada Subbab 2.5.2. Perhitungan IDF untuk *term* "properti" ditunjukkan oleh Persamaan 3.2.

$$\text{idf}_{\text{"}properti\text{"}} = \log\frac{3}{1} = 0.4771 \tag{3.2}$$

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.

Tabel 3.3: Hasil perhitungan IDF

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

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.

Tabel 3.4: Hasil perhitungan bobot TF-IDF

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

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. Meskipun *centroid* menyimpan informasi mengenai isi dari dokumen, namun *centroid* bukanlah sebuah dokumen. *Centroid* hanyalah suatu vektor pada bidang N dimensi yang merupakan titik tengah dari suatu *cluster*. Secara umum representasi *centroid* ke dalam kromosom ditunjukkan pada Gambar 3.1.

$$\begin{array}{cccc} C_1 & C_2 & & C_k \\ w_{1,1}, w_{1,2}, ..., w_{1,n}, & w_{2,1}, w_{2,2}, ..., w_{2,n} & ... & w_{k,1}, w_{k,2}, ..., w_{k,n} \end{array}$$

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 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* dihitung menggunakan *cosine similarity* seperti yang sudah dijelaskan pada Subbab 2.4. Sesuai dengan Persamaan 2.10 pada Subbab 2.6, perhitungan *fitness* sebagai metrik yang menggambarkan seberapa baiknya suatu calon solusi ditunjukkan dalam Persamaan 3.6 dengan beberapa penyesuaian.

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

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* dihitung dengan menggunakan *cosine similarity* sehingga persamaan untuk menghitung *cosine similarity* (Persamaan 2.5 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 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 digunakan.

3.6.1 Inisialisasi Populasi

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

3.6.2 Seleksi

Proses seleksi dilakukan setiap iterasi (setiap generasi) untuk memilih calon induk dari generasi selanjutnya. Pada penelitian ini 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 memiliki probabilitas lebih tinggi untuk terpilih menjadi induk dari generasi selanjutnya dan masuk ke dalam proses persilangan. Namun karena masih ada peluang individu dengan nilai *fitness* tertinggi tidak terpilih, maka dalam penelitian ini juga digunakan teknik elitisme [13]. Dengan digunakannya elitisme, maka individu dengan nilai *fitness* terbaik disalin dan langsung menjadi anggota dari generasi berikutnya. Hal ini dilakukan untuk menjamin individu terbaik tidak hilang akibat tidak terpilih oleh *roulette-wheel selection*.

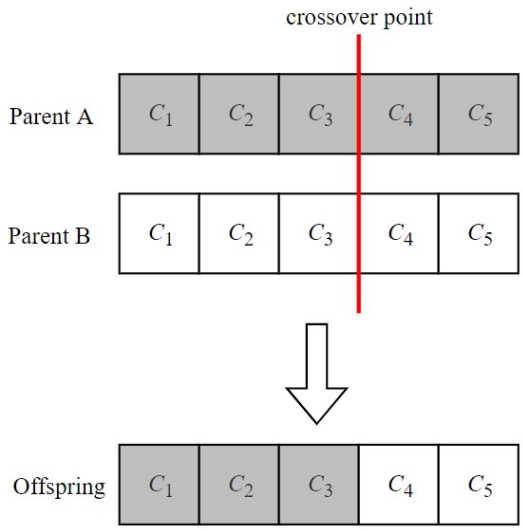
3.6.3 Persilangan

Dua kromosom yang terpilih dalam proeses seleksi 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 mengalami persilangan adalah kromosom A dengan kromosom B seperti pada Gambar 3.4. Setelah itu 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 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 mengalami mutasi dengan mengambilnya secara acak. Nilai gen yang baru dapat ditentukan menggunakan Persamaan 3.7.

$$v' = v \pm 2 * \delta * v \tag{3.7}$$

dengan v' nilai yang baru dari gen setelah mutasi, v merupakan nilai gen sebelumnya, dan δ merupakan suatu bilangan acak antara 0 sampai 1 yang dibangkitkan menggunakan pesebaran *uniform*. 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). Setelah dilakukan perhitungan nilai menggunakan Persamaan 3.7, 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	**2.59**	15.85	14.23	5.12	9.45

(c) Kromosom hasil mutasi

Gambar 3.5: Ilustrasi mutasi kromosom

3.7 Evaluasi Hasil Pengelompokan Menggunakan *Purity*

Seperti yang telah dijelaskan pada Subbab 3.1, *dataset* yang digunakan merupakan *dataset* berlabel. Untuk mengukur akurasi pengelompokan berdasarkan labelnya, maka pada penelitian ini digunakan pengukuran nilai *purity* (Subbab 2.7). Oleh karena nilai *purity* didapatkan dari membandingkan hasil pengelompokan dengan label *dataset*, maka nilai *purity* ini dapat menjadi acuan kualitas dari hasil suatu pengelompokan. Selain itu, nilai *purity* juga dapat menentukan kinerja dari metrik yang digunakan dalam proses pengelompokan.

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 diakomodasi oleh antarmuka pengguna (Graphical User Interface). Rincian masukan melalui antarmuka pengguna 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 keluaran yang ditulis pada *file* CSV dapat dilihat pada Tabel 4.1.

Format penamaan dari *file* ini adalah "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".

Tabel 4.1: Contoh keluaran dalam bentuk *file 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:

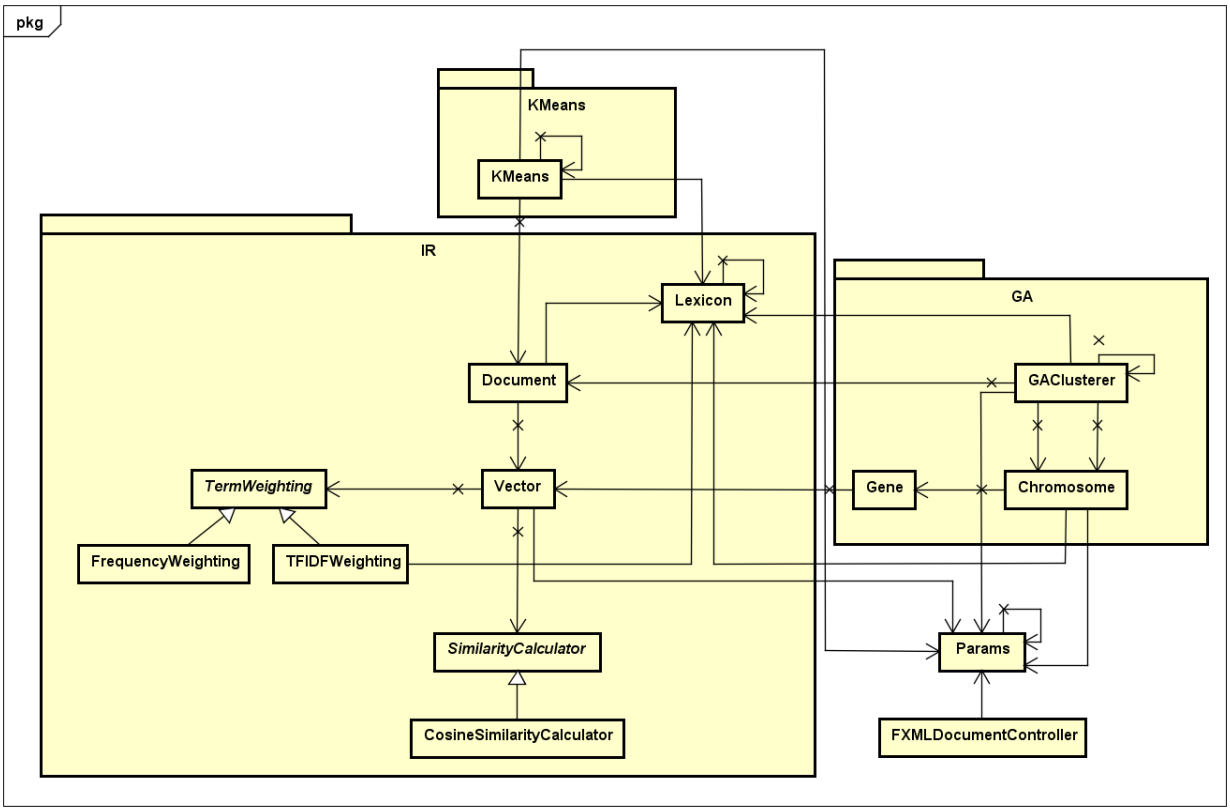
Waktu	20 menit 6 detik
Intracluster	565.9675341
Banyak Iterasi	4

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
...	...	...	...	...

4.2 Rancangan Kelas

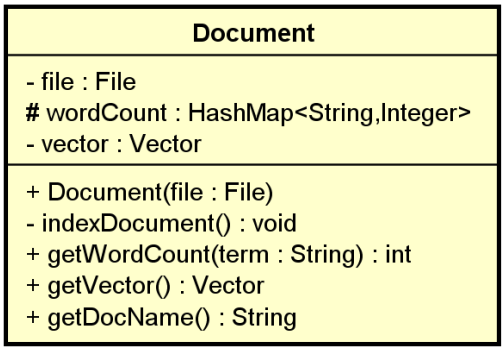
Berdasarkan hasil analisis dari masalah yang dihadapi, dibentuklah diagram kelas pada Gambar 4.1 sebagai gambaran dari perangkat lunak yang dibuat.



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. Diagram kelas pada Gambar 4.1 dibagi ke dalam tiga *package* berdasarkan fungsinya. *Package* IR berfungsi untuk mengambil informasi dari dokumen input. *Package* GA berfungsi untuk melakukan pengelompokan dokumen dengan menggunakan algoritma genetika. Sedangkan *package* *K-means* berfungsi untuk melakukan pengelompokan dokumen menggunakan algoritma *K-means*. Beberapa kelas dalam penelitian ini dibuat sebagai kelas *singleton* untuk menjaga integritas data yang disimpan saat ada lebih dari satu *instance* yang ingin mengakses atribut-atributnya. Penjelasan dari setiap kelas dalam Gambar 4.1 adalah sebagai berikut.

4.2.1 *Document*



Gambar 4.2: Kelas *Document*

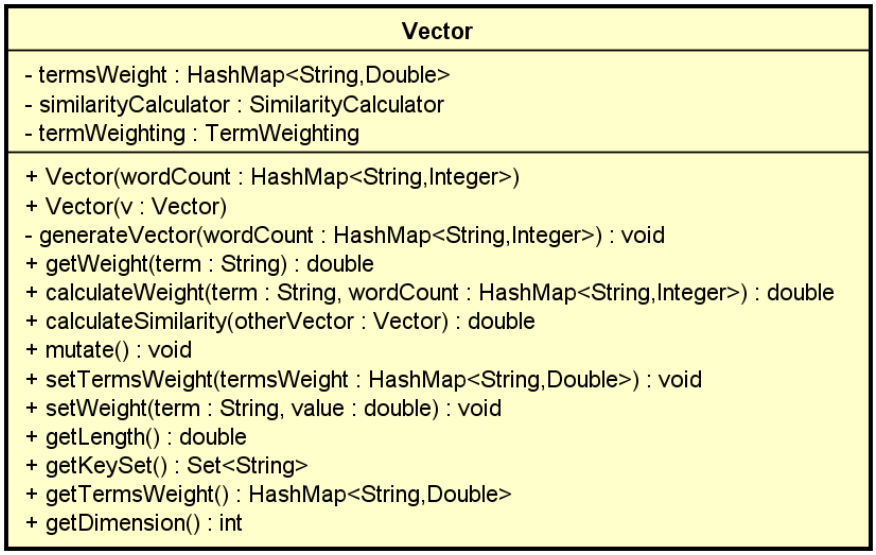
Kelas ini merupakan representasi dari dokumen yang diproses dalam pengelompokan. Kelas ini berfungsi untuk menyimpan informasi yang dibutuhkan dari sebuah dokumen selama proses pengelompokan. Atribut yang dimiliki oleh kelas *Document* adalah:

- *file*: atribut ini bertipe *File* milik *package* java.io yang berfungsi untuk merepresentasikan *file* dari dokumen yang 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 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:

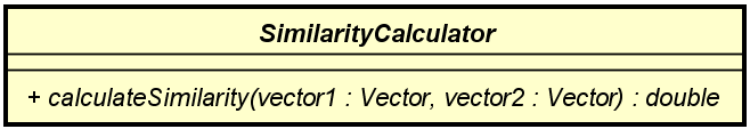
- *termsWeight*: atribut ini bertipe *Hashmap* dengan *key* berupa *String* dan *value* berupa *Double*. Atribut ini menyimpan pasangan *term* dan bobotnya sesuai dengan metode pembobotan.
- *similarityCalculator*: atribut ini bertipe *SimilarityCalculator* dan merupakan objek yang digunakan untuk menghitung kemiripan antar vektor.
- *termWeighting*: atribut ini bertipe *TermWeighting* dan merupakan objek yang 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.
- *Vector*: *method* ini merupakan constructor dengan sebuah parameter yaitu objek *Vector* yang berfungsi untuk menduplikasi objek *Vector*.
- *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.

- *mutate*: *method* ini berfungsi untuk melakukan operasi mutasi pada tingkat vektor.
- *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.

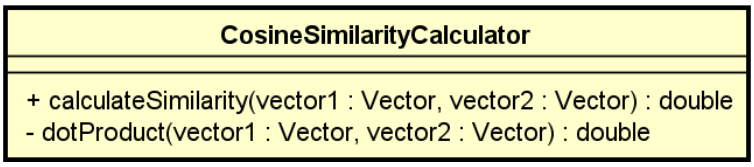
4.2.3 *SimilarityCalculator*



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*

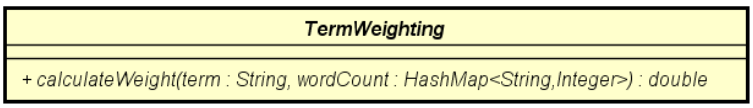


Gambar 4.5: Kelas *CosineSimilarityCalculator*

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 2.5).
- *dotProduct*: *method* ini berfungsi untuk menghitung hasil perkalian titik (*dot product*) antara *vector1* dan *vector2*.

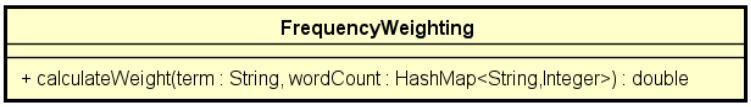
4.2.5 *TermWeighting*



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.

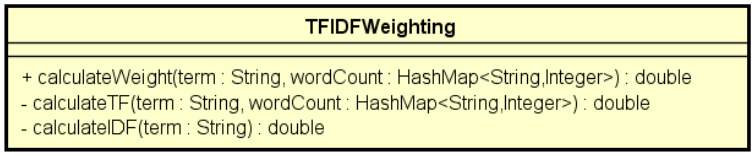
4.2.6 *FrequencyWeighting*



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*

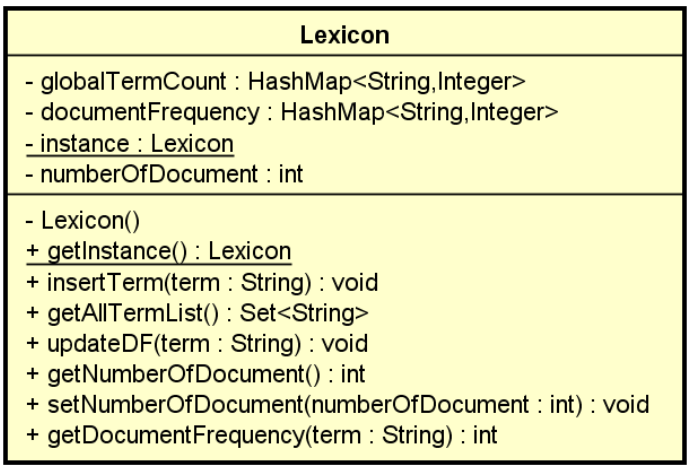


Gambar 4.8: Kelas *TFIDFWeighting*

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*

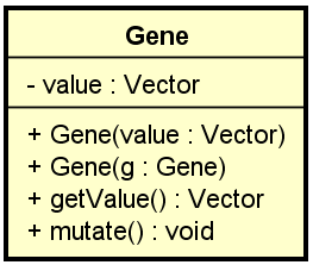
Kelas ini merepresentasikan sebuah kamus yang menangani seluruh kebutuhan dalam proses pengelompokan yang membutuhkan akses global untuk keseluruhan koleksi dokumen. Atribut yang ada dalam kelas ini adalah:

- *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*

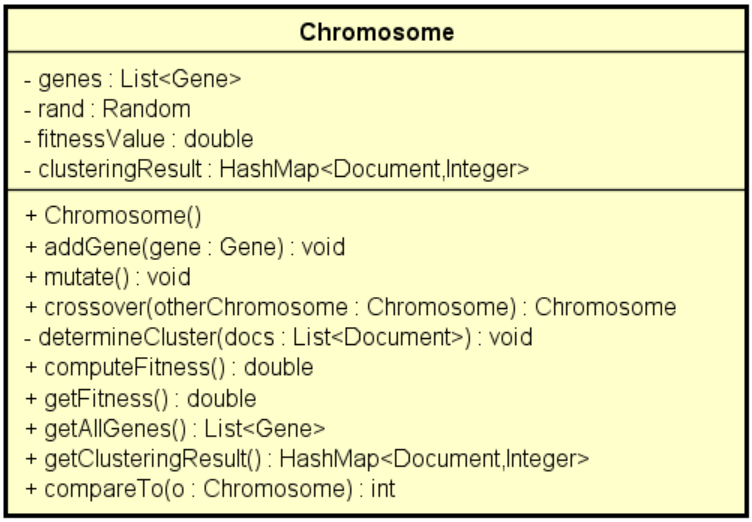


Gambar 4.10: Kelas *Gene*

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*.
- *Gene*: *method* ini merupakan *overloading constructor* yang berfungsi untuk menduplikasi objek dari kelas *Gene*.
- *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. Atribut yang terdapat dalam kelas ini adalah:

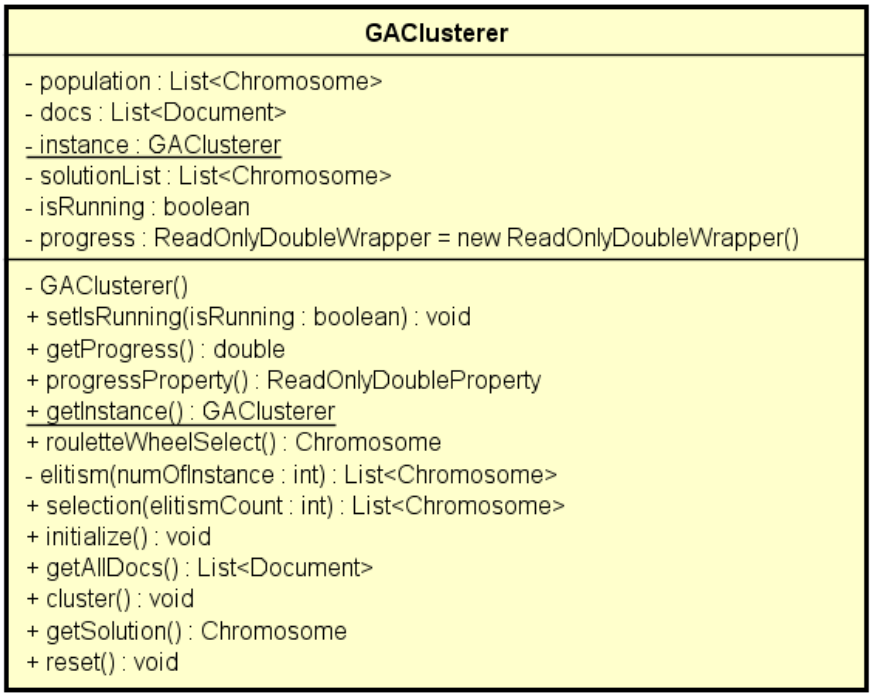
- *genes*: atribut bertipe *List of Gene* dan merupakan kumpulan gen yang terdapat dalam kromosom.

- *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 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 2.3.1).
- *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 digunakan dalam proses *sorting*.

4.2.11 *GAClusterer*



Gambar 4.12: Kelas *GAClusterer*

Kelas ini merupakan kelas utama yang 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 langsung masuk ke generasi berikutnya. Cara kerja *method* ini dijelaskan dalam Algoritma 6.

Algoritma 6 Elitisme Algoritma Genetika

function ELITISME(*count*, *populasi*) **returns** individu-individu elit

inputs: *count*, banyaknya individu elit
populasi, himpunan individu

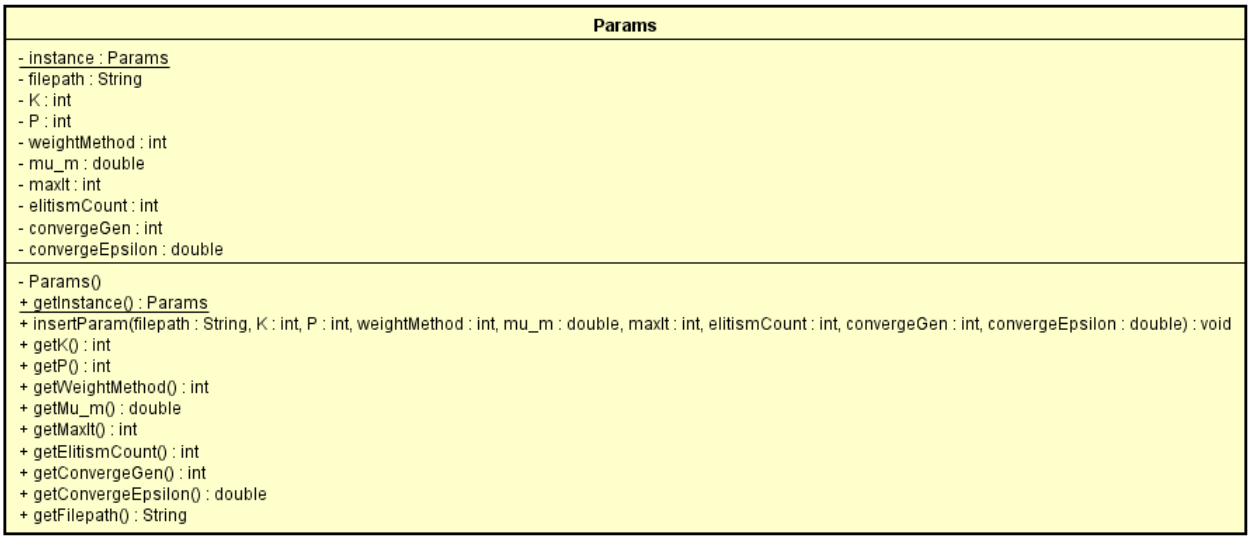
```
1: pq ← Priority Queue
2: for i=1 to SIZE(populasi) do
3:    if count == 0  then
4:       break
5:    end if
6:    if i < count  then
7:       OFFER(pq, populasi[i])
8:    else
9:       if FITNESS(populasi[i]) > FITNESS(PEEK(pq)) then
10:         POLL(pq)
11:         OFFER(pq,populasi[i])
12:      end if
13:   end if
14: end for
15: return TO-ARRAY(pq)
```

Penjelasan untuk fungsi Elitisme adalah sebagai berikut:

- Individu elit akan dipilih dengan cara mengiterasi seluruh *populasi* (baris 2)
- Apabila jumlah elit yang diinginkan adalah 0 ($count == 0$), maka iterasi akan diberhentikan (baris 3 dan 4).
- Apabila banyaknya individu dalam *pq* masih kurang dari *count* (baris 6), maka akan dilakukan pemanggilan fungsi *OFFER* yang berfungsi untuk memasukkan individu ke-i dari populasi ke dalam *pq* (baris 7).
- Pada baris 9, fungsi *PEEK* berfungsi untuk mengembalikan elemen paling awal dalam *pq*. Kemudian akan dibandingkan nilai *fitness* antara elemen paling awal *pq* dan individu ke-i dalam populasi.
- Apabila nilai *fitness* individu ke-i dalam populasi lebih besar daripada elemen paling awal dalam *pq*, maka akan dilakukan dua hal dalam baris 10 dan 11.
- Pada baris 10, fungsi *POLL* berfungsi untuk mengeluarkan elemen paling awal dalam *pq*.
- Pada baris 11, fungsi *OFFER* akan memasukkan individu ke-i dari populasi ke dalam *pq*.
- Pada baris 15, fungsi *TO-ARRAY* akan mengubah *pq* ke dalam bentuk *array*.

- *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 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 dikelompokkan.
- *K*: atribut ini berfungsi untuk menyimpan banyaknya *cluster* yang dipentuk dalam proses pengelompokan.
- *P*: atribut ini berfungsi untuk menyimpan banyaknya populasi yang dibentuk dalam proses pengelompokan menggunakan algoritma genetika.
- *weightingMethod*: atribut ini berfungsi untuk menyimpan metode pembobotan yang digunakan dalam proses pengelompokan. Apabila atribut ini bernilai 0 maka metode yang digunakan adalah TF-IDF sedangkan apabila bernilai 1 maka metode yang 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 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 digunakan untuk membandingkan *fitness* tiap solusi dari setiap generasi untuk menentukan apakah sudah tercapai konvergen atau belum.

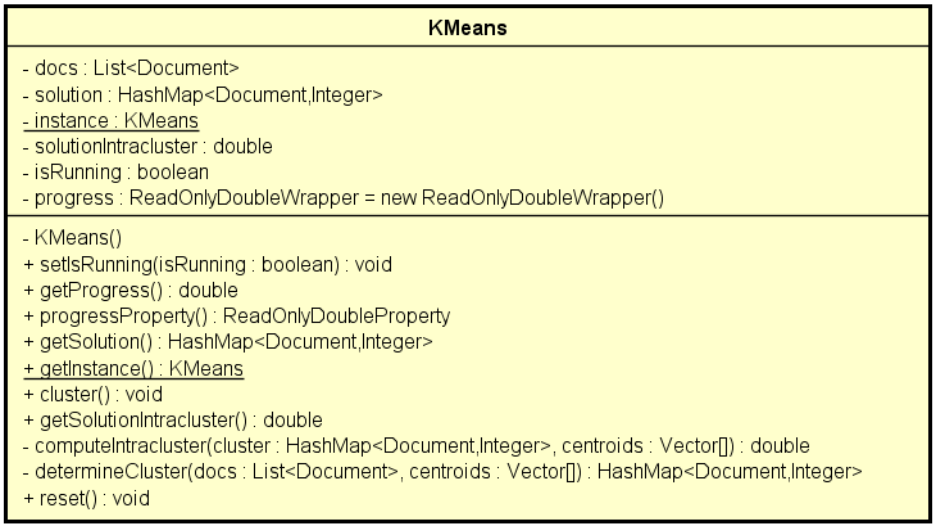
Method yang ada pada kelas ini adalah:

- *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*

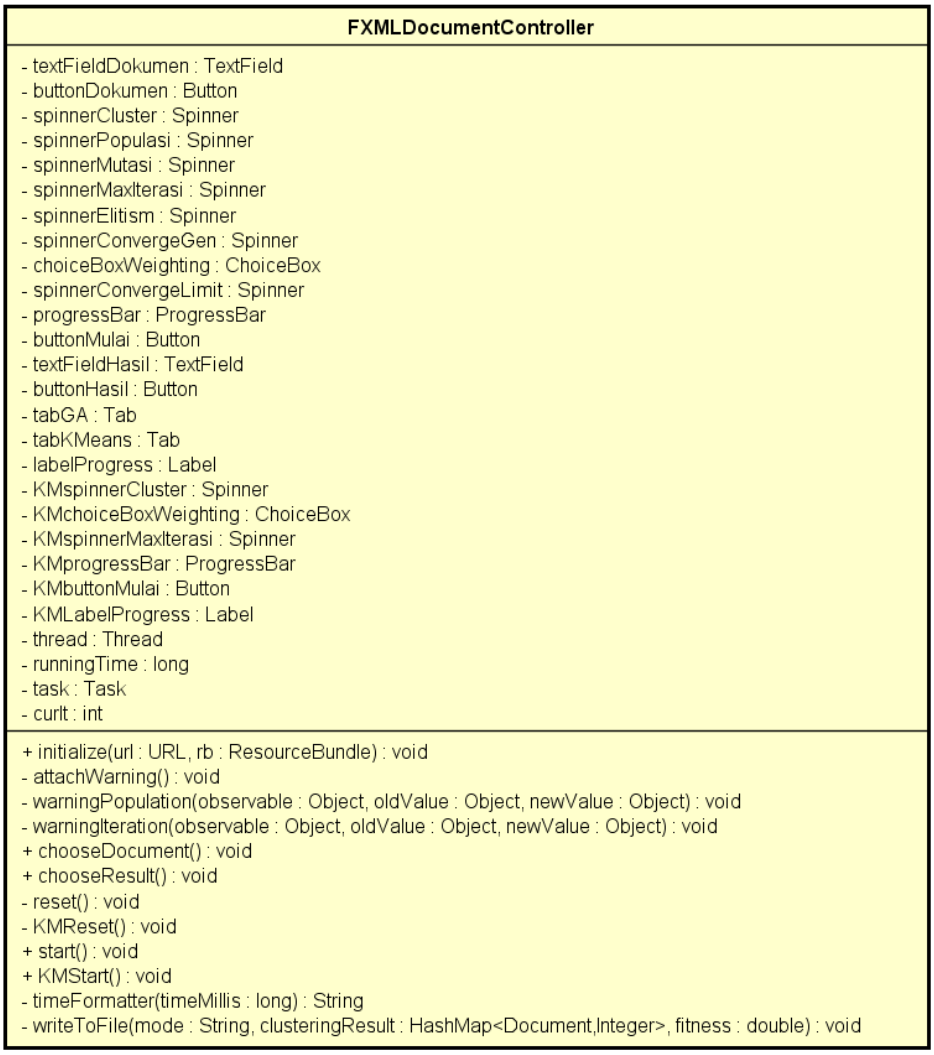
Kelas ini merupakan kelas utama yang mengatur jalannya proses pengelompokan menggunakan algoritma *K-means*. Kelas ini merupakan kelas *singleton*. Atribut yang terdapat dalam kelas ini adalah:

- *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 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.
- *reset*: *method* ini berfungsi untuk mengatur ulang seluruh atribut untuk proses pengelompokan berikutnya.

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 menangani masukan untuk parameter banyaknya *cluster* pada antarmuka pengguna di bagian GA.
- *spinnerPopulasi*: atribut ini bertipe *Spinner* yang menangani masukan untuk parameter banyaknya populasi pada antarmuka pengguna di bagian GA.
- *spinnerMutasi*: atribut ini bertipe *Spinner* yang menangani masukan untuk parameter probabilitas mutasi pada antarmuka pengguna di bagian GA.

- *spinnerMaxIterasi*: atribut ini bertipe *Spinner* yang menangani masukan untuk parameter maksimum iterasi pada antarmuka pengguna di bagian GA.
- *spinnerElitism*: atribut ini bertipe *Spinner* yang menangani masukan untuk parameter individu elitisme pada antarmuka pengguna di bagian GA.
- *spinnerConvergeGen*: atribut ini bertipe *Spinner* yang menangani masukan untuk parameter banyaknya generasi konvergen pada antarmuka pengguna di bagian GA.
- *choiceBoxWeighting*: atribut ini bertipe *ChoiceBox* yang menangani masukan untuk parameter metode pembobotan pada antarmuka pengguna di bagian GA.
- *spinnerConvergeLimit*: atribut ini bertipe *Spinner* yang menangani masukan untuk parameter banyaknya generasi konvergen pada antarmuka pengguna di bagian GA.
- *progressBar*: atribut ini bertipe *progressBar* yang 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 menangani masukan untuk parameter banyaknya *cluster* pada antarmuka pengguna di bagian *K-means*.
- *KMchoiceBoxWeighting*: atribut ini bertipe *ChoiceBox* yang menangani masukan untuk parameter metode pembobotan pada antarmuka pengguna di bagian *K-means*.
- *KMspinnerMaxIterasi*: atribut ini bertipe *Spinner* yang menangani masukan untuk parameter maksimum iterasi pada antarmuka pengguna di bagian *K-means*.
- *KMprogressBar*: atribut ini bertipe *progressBar* yang 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 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.

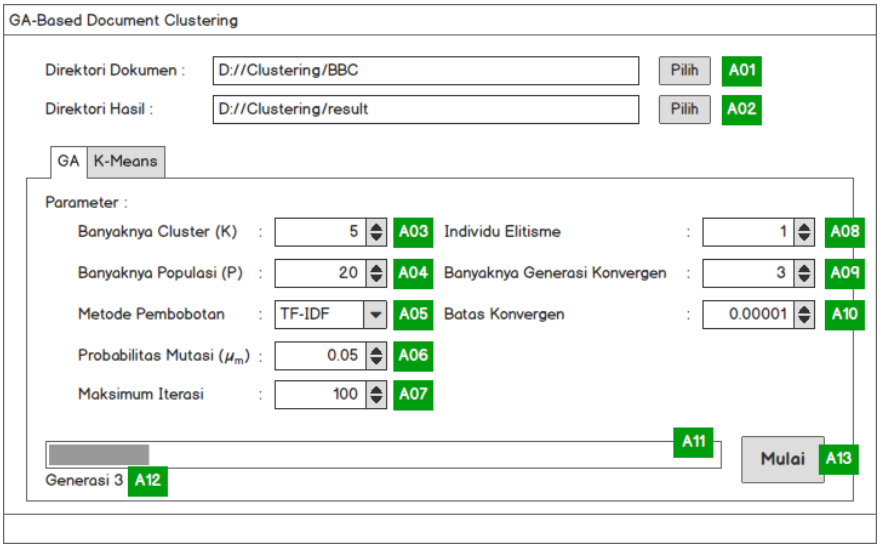
4.3 Perancangan Antarmuka Pengguna

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* diberi label unik agar dapat disesuaikan dengan tabel keterangan. Berikut dibahas rancangan antarmuka pengguna dari perangkat ini.

[1] https://balsamiq.com/

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.



Gambar 4.16: Rancangan antarmuka halaman algoritma genetika

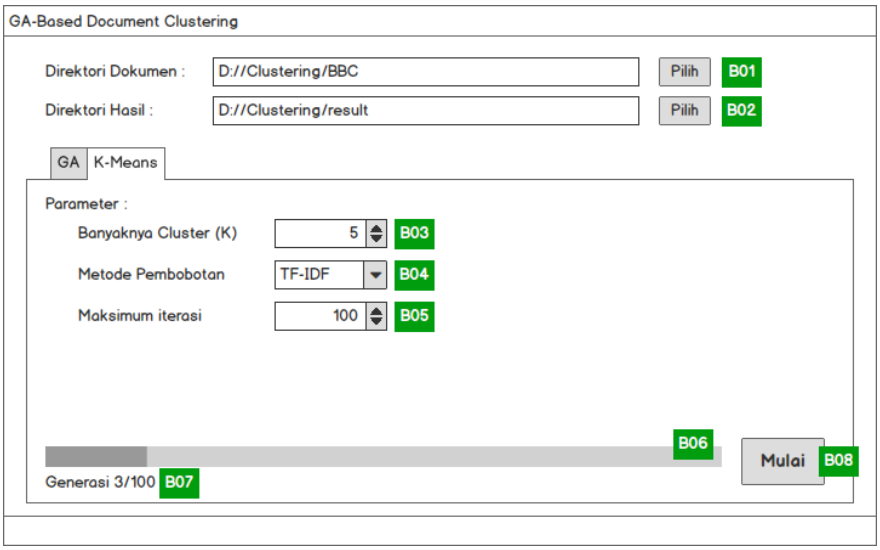
Penjelasan setiap *field* dalam halaman ini dijelaskan dalam Tabel 4.2.

Tabel 4.2: Rincian *field* pada halaman algoritma genetika

Kode	**Nama**	**Jenis**	***Defaut value***	**Wajib**	**Keterangan**
A01	Direktori Dokumen	*file chooser*	-	ya	Folder yang dipilih harus berisi file teks yang akan dikelompokan menggunakan perangkat lunak
A02	Direktori Hasil	*file chooser*	-	ya	Folder yang dipilih merupakan folder tempat menyimpan file CSV hasil pengelompokan
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	Hanya bisa bernilai "TF-IDF" dan "Frekuensi"
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}

Objek dengan kode A11 merupakan sebuah *progress bar* yang menampilkan perkembangan dari jalannya program untuk setiap iterasi. Apabila proses dalam suatu iterasi sudah selesai, maka 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 memiliki nilai apabila program belum dijalankan dan menampilkan status "Inisialisasi..." apabila program sedang melakukan pengindeksan dokumen dan inisialisasi populasi. Setelah iterasi dimulai maka label A12 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 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.

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

Kode	**Nama**	**Jenis**	***Defaut value***	**Wajib**	**Keterangan**
B01	Direktori Dokumen	*file chooser*	-	ya	Folder yang dipilih harus berisi file teks yang akan dikelompokan menggunakan perangkat lunak
B02	Direktori Hasil	*file chooser*	-	ya	Folder yang dipilih merupakan folder tempat menyimpan file CSV hasil pengelompokan
B03	Banyaknya Cluster	*spinner*	5	ya	Nilai minimum 1
B04	Metode Pembobotan	*dropdown*	TF-IDF	ya	Hanya bisa bernilai "TF-IDF" dan "Frekuensi"
B05	Maksimum Iterasi	*spinner*	100	ya	Nilai minimum 1

Dalam Tabel 4.3, objek dengan kode B06 merupakan sebuah *progress bar* yang menampilkan perkembangan dari jalannya program yang menyatakan banyaknya iterasi yang sudah selesai dijalankan dibandingkan dengan maksimum iterasi. Label B07 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 diteliti di antaranya 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 menjadi variabel bebas. Parameter masukan yang merupakan variabel bebas pada penelitian ini di antaranya adalah:

1. Banyaknya populasi
   Jumlah individu yang dibentuk dalam proses pengelompokan menggunakan algoritma genetika.
2. Metode pembobotan
   Metode penghitungan bobot suatu *term* dalam vektor. Dalam penelitian ini digunakan dua metode yaitu bobot TF-IDF dan bobot frekuensi.
3. Probabilitas mutasi
   Probabilitas terjadinya mutasi pada saat proses pembentukan keturunan dalam algoritma genetika.
4. Individu elitisme
   Jumlah individu dengan *fitness* terbaik yang langsung masuk ke generasi selanjutnya.

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

- Banyaknya *cluster*
  Banyaknya cluster yang ingin dibentuk pada saat proses pengelompokan.
- Maksimum iterasi
  Batas maksimum iterasi yang dapat dilakukan selama proses pengelompokan dokumen.
- Banyaknya generasi konvergen
  Banyaknya generasi terakhir yang dilihat selisih nilai *fitness*-nya.

- Batas konvergen
  Batas selisih nilai *fitness* supaya dianggap memiliki nilai *fitness* sama.

Variabel banyaknya *cluster* 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
  Waktu yang diperlukan untuk mengelompokkan data.
- Nilai *intracluster*
  Jumlah *similarity* antara setiap dokumen ke masing-masing *centroid*.
- Banyak iterasi
  Jumlah iterasi yang dilakukan selama proses pengelompokan.
- Nilai *purity*
  Nilai yang menunjukkan akurasi dari pengelompokan (Subbab 2.7).

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, dibuat suatu skenario pengujian eksperimental untuk mengetahui pengaruh variabel bebas terhadap variabel terikat. Variasi nilai dari variabel bebas ditunjukkan dalam Tabel 5.1.

Tabel 5.1: Variasi nilai variabel bebas

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

Setiap variabel dalam Tabel 5.1 memiliki dua sampai tiga variasi nilai. Karena keterbatasan waktu, maka pengujian eksperimental ini tidak menguji seluruh kombinasi parameter yang ada. Variasi dari parameter dalam Tabel 5.1 hanya menggantikan nilai pada suatu kasus uji yang disebut kasus uji ideal. Kasus uji ideal merupakan kasus uji yang dianggap menghasilkan keluaran yang ideal. Kasus uji ideal untuk pengujian eksperimental dalam penelitian ini (menggunakan dataset yang telah dibahas pada Subbab 3.1) adalah sebagai berikut:

- 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: 0.00001

5.2 Eksperimen Algoritma Genetika

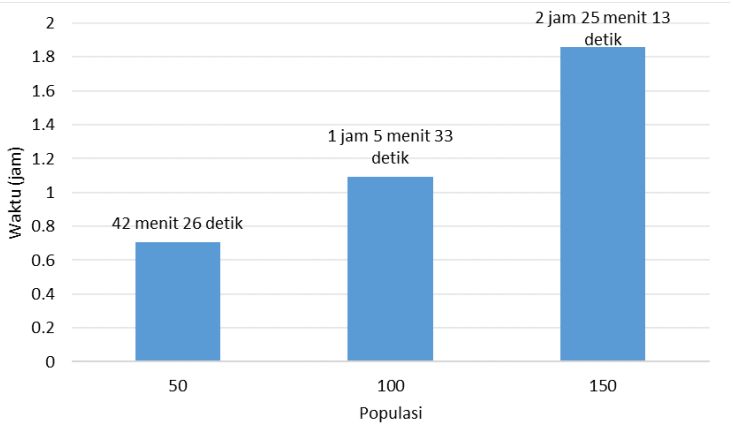
Eksperimen dibagi menjadi beberapa bagian berdasarkan parameter yang diubah. Terdapat 4 bagian yaitu berdasarkan keempat variabel bebas yang telah dijelaskan pada Subbab 5.1. Untuk setiap variasi parameter, dilakukan lima kali pengujian dan dihitung rata-rata hasilnya. Berikut adalah hasil dari eksperimen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

1. Banyaknya populasi:

   Tabel 5.2: Rata-rata hasil pengelompokan dengan variasi variabel 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

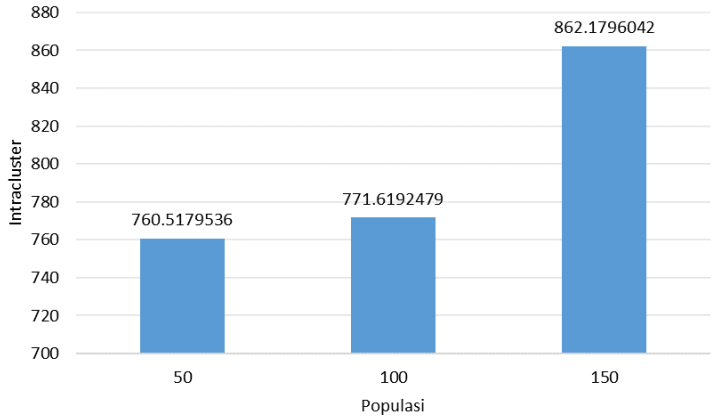
   Berdasarkan Tabel 5.2, Kenaikan populasi berbanding lurus dengan kenaikan waktu pemrosesan dan *intracluster similarity*. Sedangkan untuk banyak iterasi dan nilai *purity* sama sekali tidak bergantung dengan kenaikan populasi. Dapat disimpulkan bahwa kenaikan populasi tidak mempengaruhi hasil pengelompokan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



Gambar 5.1: Grafik hubungan banyaknya populasi dengan waktu pengelompokan

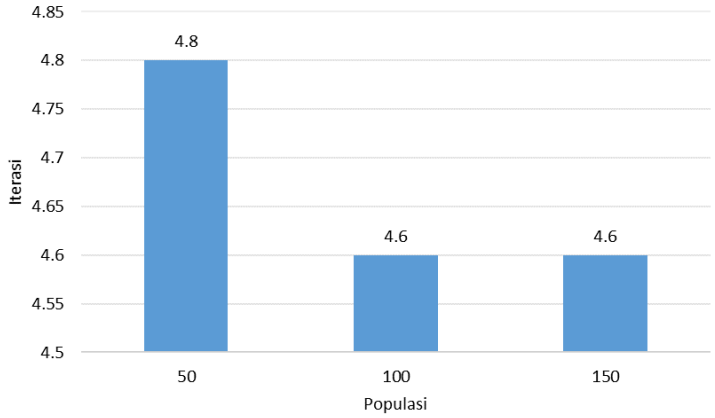
Dapat disimpulkan dari diagram pada Gambar 5.1, pada saat populasi berjumlah 100, waktu tempuh lebih lama 55% dari saat populasi berjumlah 50. Pada saat populasi berjumlah 150, waktu tempuh 70% lebih lama dibandingkan denga waktu pada saat populasi berjumlah 100.

Kompleksitas algoritma genetika tidaklah linear sehingga menyebabkan kenaikan waktunya juga tidak linear. Selain itu, semakin banyak populasi maka calon solusi semakin banyak pula. Hal ini menyebabkan GA lebih sulit mencapai konvergen.



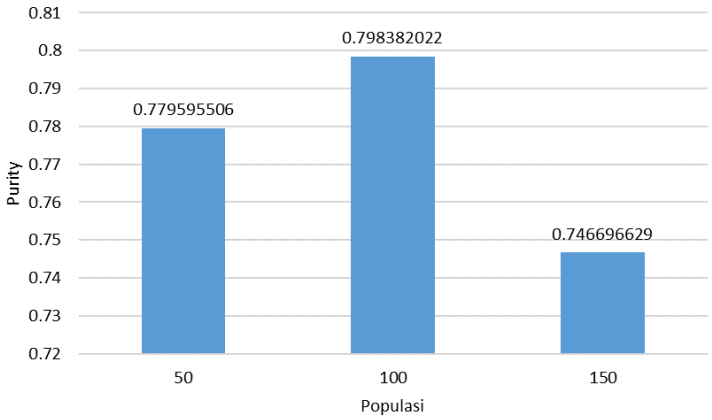
Gambar 5.2: Diagram hubungan banyaknya populasi dengan *intracluster similarity*

Dapat disimpulkan dari diagram pada Gambar 5.2, kenaikan populasi juga mempengaruhi kenaikan *intracluster similarity*. *Intracluster similarity* pada saat populasi berjumlah 100 lebih besar 1% dibandingkan dengan *intracluster similarity* pada saat populasi berjumlah 50. Sedangkan *intracluster similarity* pada saat populasi berjumlah 150 lebih besar 11% dibandingkan dengan *intracluster similarity* pada saat populasi berjumlah 100. Hal ini dikarenakan dengan populasi yang lebih banyak maka kemungkinan untuk mendapat solusi yang lebih baik semakin besar.



Gambar 5.3: Diagram hubungan banyaknya populasi dengan banyaknya iterasi

Dapat disimpulkan dari diagram 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: Diagram hubungan banyaknya populasi dengan nilai *purity*

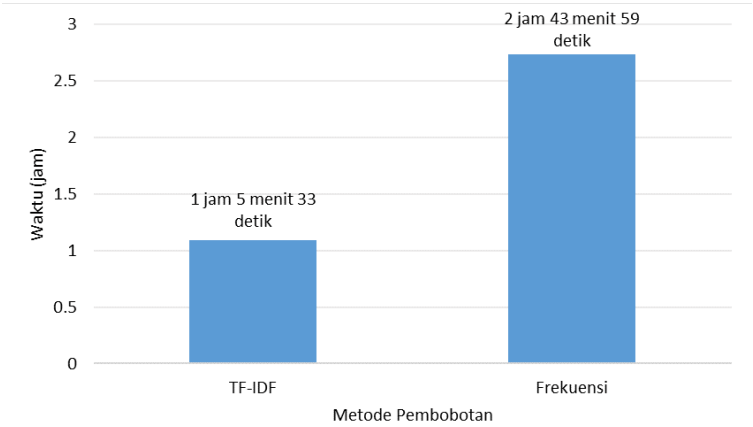
Dapat disimpulkan dari diagram pada Gambar 5.4, kenaikan populasi juga tidak mempengaruhi nilai *purity* sama seperti banyaknya iterasi. Berdasarkan nilai *purity*, maka hasil terbaik diperoleh dengan populasi sebanyak 100 individu. Nilai *purity* pada saat populasi berjumlah 100 lebih besar 2% dibandingkan dengan pada saat populasi berjumlah 50 dan 7% lebih besar dibandingkan dengan pada saat populasi berjumlah 150.

2. Metode pembobotan:

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

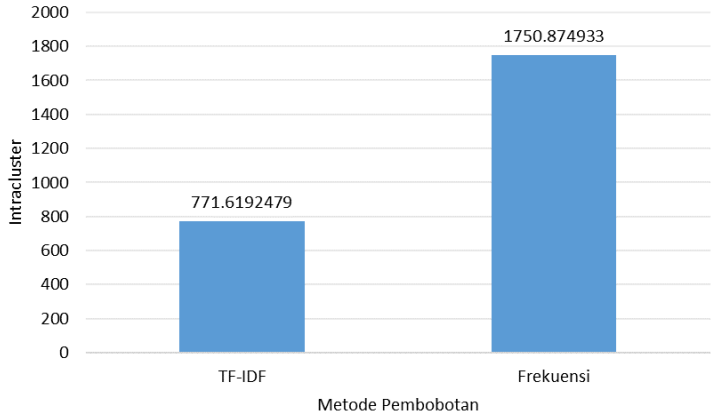
Bobot	Waktu (jam)	*Intracluster*	Iterasi	*Purity*
TF-IDF	1.092666667	771.6192479	4.6	0.798382022
Frekuensi	2.733222222	1750.874933	7	0.571325843

Berdasarkan Tabel 5.3, dibentuk empat buah diagram (Gambar 5.5 - 5.8) untuk menunjukan hubungan antara metode pembobotan dengan keempat variabel terikat.



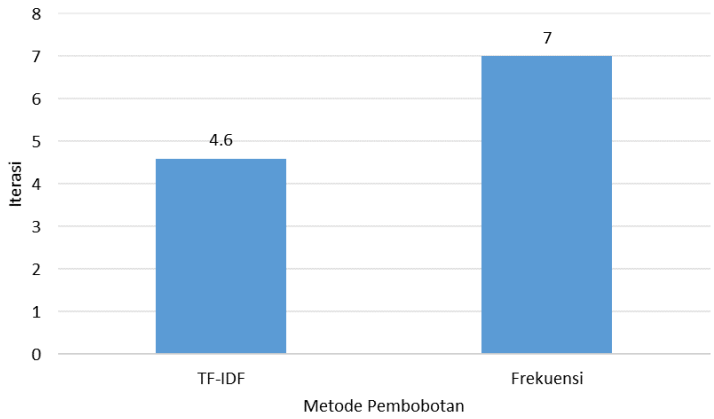
Gambar 5.5: Diagram hubungan metode pembobotan dengan waktu pengelompokan

Berdasarkan diagram pada Gambar 5.5, waktu yang diperlukan untuk pengelompokan menggunakan bobot TF-IDF jauh lebih cepat hingga 167% dibandingan dengan menggunakan bobot frekuensi. Hal ini kemungkinan memiliki kaitan dengan banyaknya iterasi yang terjadi selama proses pengelompokan.



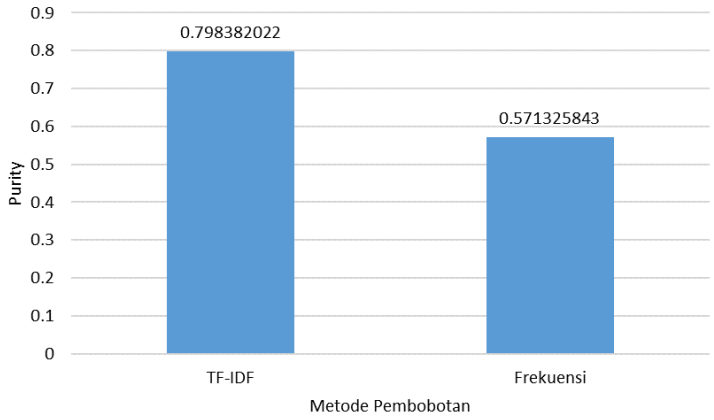
Gambar 5.6: Diagram hubungan metode pembobotan dengan *intracluster similarity*

Berdasarkan diagram pada Gambar 5.6, *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.



Gambar 5.7: Diagram hubungan metode pembobotan dengan banyaknya iterasi

Berdasarkan diagram pada Gambar 5.7, banyaknya iterasi yang dilakukan 52% lebih banyak apabila menggunakan bobot frekuensi dibandingkan dengan menggunakan bobot TF-IDF.



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

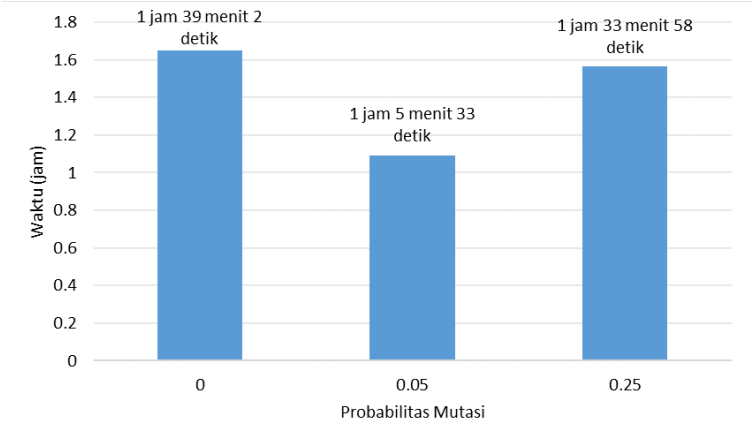
Berdasarkan diagram pada Gambar 5.8, nilai *purity* yang didapatkan apabila menggunakan bobot TF-IDF 40% lebih besar apabila dibandingkan dengan menggunakan bobot frekuensi. Hal ini terjadi karena dengan menggunakan bobot TF-IDF, representasi dokumen menjadi lebih akurat. Perhitungan bobot menggunakan TF-IDF tidak hanya ditentukan berdasarkan apa yang ada di dalam dokumen itu saja (lokal), namun juga mempertimbangkan faktor frekuensi dokumen secara global (Subbab 2.5.2).

3. Probabilitas mutasi:

Tabel 5.4: Rata-rata hasil pengelompokan dengan variasi variabel 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

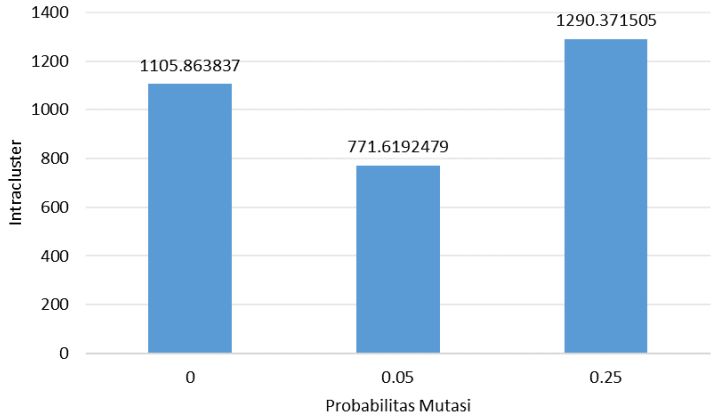
Berdasarkan Tabel 5.4, dibentuk empat buah diagram (Gambar 5.9 - 5.12) untuk menunjukan hubungan antara probabilitas mutasi dengan keempat variabel terikat.



Gambar 5.9: Diagram hubungan probabilitas mutasi dengan waktu pengelompokan

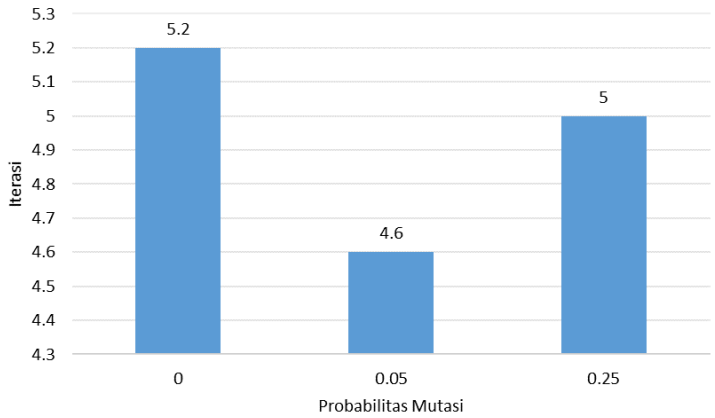
Dapat disimpulkan dari diagram pada Gambar 5.9, waktu tempuh paling cepat didapatkan dengan menggunakan probabilitas mutasi 0.05. Apabila probabilitas mutasi bernilai 0, maka

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.



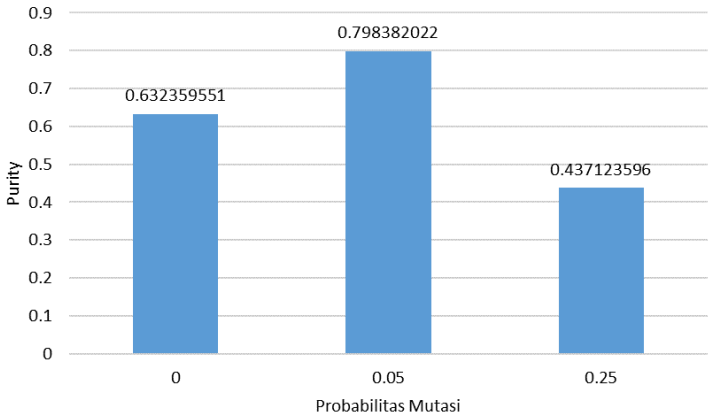
Gambar 5.10: Diagram hubungan probabilitas mutasi dengan *intracluster similarity*

Berdasarkan *intracluster similarity* pada diagram 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: Diagram hubungan probabilitas mutasi dengan banyaknya iterasi

Dapat disimpulkan dari diagram pada Gambar 5.11, jumlah iterasi paling sedikit 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 sulit mencapai konvergen karena proses pencariannya menjadi tidak terarah.



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

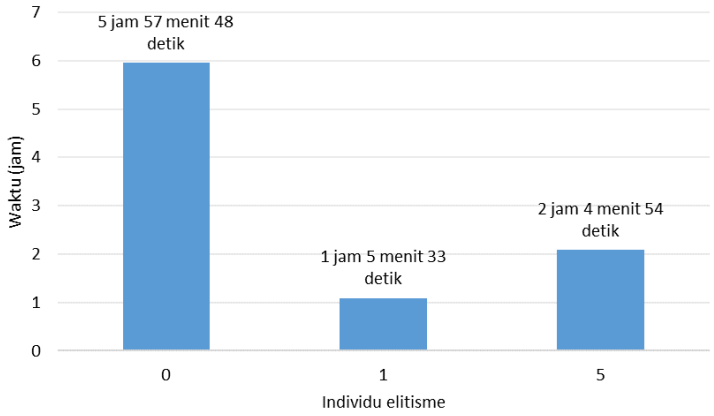
Berdasarkan nilai *purity* pada diagram dalam Gambar 5.12, hasil terbaik didapatkan dengan menggunakan probabilitas mutasi 0.05. Nilai *purity* dengan menggunakan probabilitas mutasi 0.05 lebih baik 26% dibandingkan dengan menggunakan probabilitas mutasi 0 dan lebih baik 82% dibandingkan menggunakan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

4. Individu elitisme:

Tabel 5.5: Rata-rata hasil pengelompokan dengan variasi variabel individu elitisme

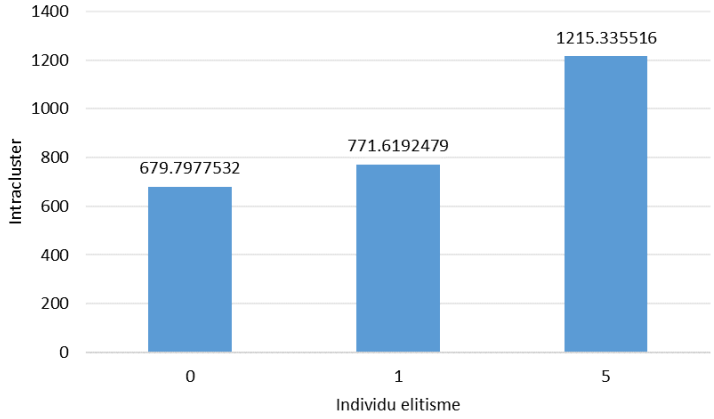
Individu elitisme	Waktu (jam)	*Intracluster*	Iterasi	*Purity*
0	5.963388889	679.7977532	15	0.826247191
1	1.092666667	771.6192479	4.6	0.798382022
5	2.081777778	1215.335516	6	0.527101124

Karena keterbatasan waktu, untuk eksperimen tanpa individu elitisme (individu elitisme=0), banyaknya iterasi maksimum hanya dibatasi sampai dengan 15 iterasi saja. Berdasarkan Tabel 5.5, dibentuk empat buah diagram (Gambar 5.13 - 5.16) untuk menunjukan hubungan antara metode pembobotan dengan keempat variabel terikat.



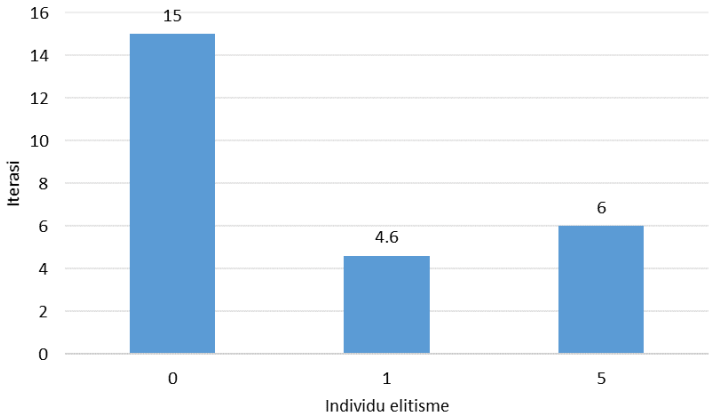
Gambar 5.13: Diagram hubungan individu elitisme dengan waktu pengelompokan

Dapat disimpulkan dari diagram pada Gambar 5.13, waktu tempuh pada saat individu elitisme berjumlah 1 paling sebentar apabila dibandingkan dengan pada saat individu elitisme berjumlah 0 atau 5. Waktu terlama dicapai pada saat tidak ada individu elitisme. Hal ini dikarenakan algoritma genetika sulit mencapai konvergen tanpa bantuan elitisme. Apabila jumlah individu elitisme terlalu banyak (pada eksperimen ini berjumlah 5), calon solusi dengan nilai *fitness* tinggi lebih banyak sehingga diperlukan waktu yang lebih lama untuk dapat mencapai konvergen.



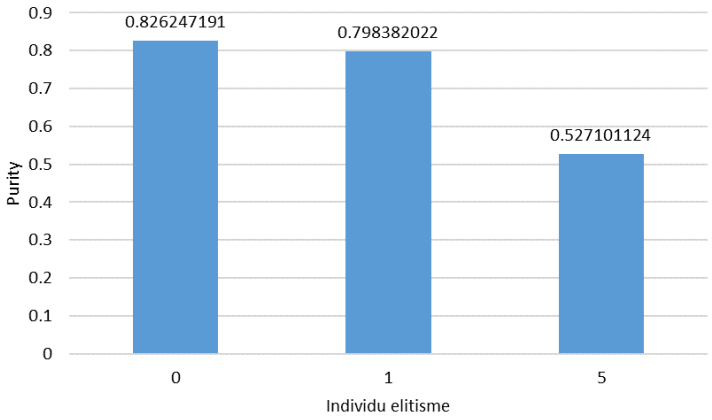
Gambar 5.14: Diagram hubungan individu elitisme dengan *intracluster similarity*

Dapat disimpulkan dari diagram pada Gambar 5.14, semakin banyak jumlah individu elitisme, maka semakin banyak individu dengan nilai *fitness* tinggi yang dipertahankan sehingga memperbesar kemungkinan untuk menghasilkan individu dengan nilai *fitness* yang lebih tinggi lagi.



Gambar 5.15: Diagram hubungan individu elitisme dengan banyaknya iterasi

Sama halnya dengan waktu, berdasarkan diagram pada Gambar 5.15, banyaknya iterasi pada saat proses pengelompokan terjadi tanpa individu elitisme lebih banyak (dalam eksperimen ini mencapai batas maksimum iterasi). Hal ini terjadi karena lebih sulit untuk mencapai konvergen tanpa ada bantuan dari elitisme. Banyaknya iterasi paling minimum dicapai dengan 1 individu elitisme. Proses pengelompokan dengan banyak individu elitisme hanya memperlambat proses pengelompokan karena kandidat solusi dengan nilai *fitness* tinggi menjadi lebih banyak dan terus dipertahankan dalam populasi sehingga memperlambat GA dalam mencapai konvergen.



Gambar 5.16: Diagram hubungan individu elitisme dengan nilai *purity*

Dapat disimpulkan dari diagram pada Gambar 5.16, nilai *purity* pada saat individu elitisme berjumlah 0 lebih besar 3% dibandingkan dengan pada saat individu elitisme berjumlah 1. Hal ini mungkin disebabkan oleh banyaknya iterasi yang ditempuh pada saat individu elitisme berjumlah 0 jauh lebih banyak dibandingkan dengan pada saat individu elitisme berjumlah 1. Dengan jumlah iterasi yang lebih banyak, maka hasilnya semakin baik. Nilai *purity* pada saat individu elitisme berjumlah 1 lebih besar 51% dibandingkan dengan pada saat individu elitisme berjumlah 5. Hal ini mungkin terjadi karena individu yang dibawa pada saat proses elitisme mungkin saja individu dengan nilai *fitness* rendah. Semakin banyak individu yang

dibawa ke generasi selanjutnya dalam elitisme, maka semakin besar kemungkinan membawa individu dengan nilai *fitness* rendah.

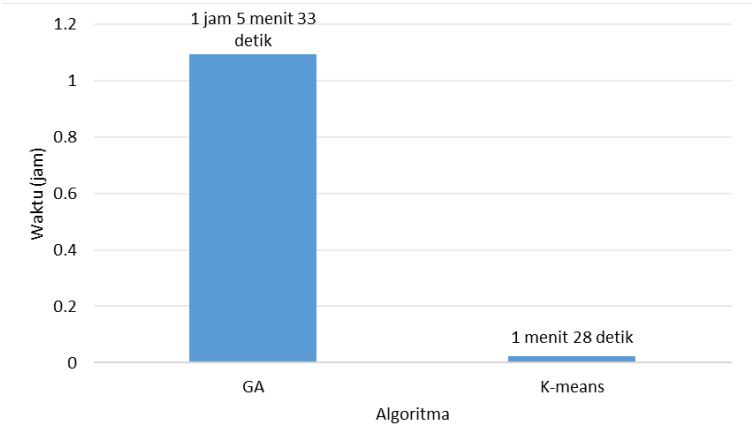
5.3 Eksperimen *K-Means*

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 di antara 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 diuji. Hasil dari eksperimen menggunakan algoritma *K-means* ini kemudian dibandingkan dengan kasus uji ideal pada algoritma genetika. Berbeda dari pengujian menggunakan algoritma genetika, pengujian menggunakan *K-means* dilakukan sebanyak 10 kali tiap variasi. Hasil eksperimen menggunakan algoritma *K-means* dijelaskan dalam Tabel 5.6.

Tabel 5.6: Rata-rata hasil pengelompokan dengan menggunakan algoritma *K-means*

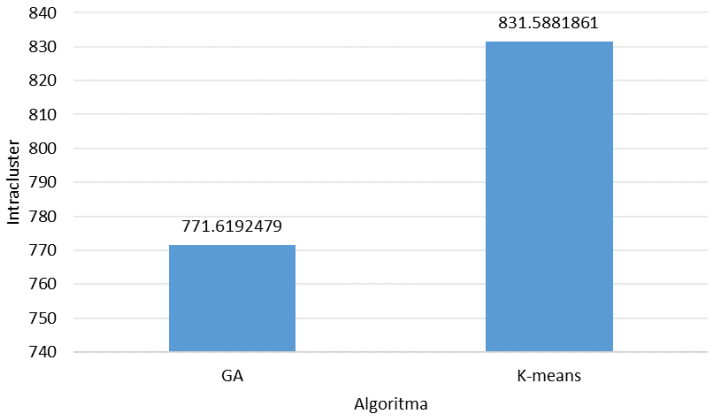
Algoritma	Waktu (jam)	*Intracluster*	Iterasi	*Purity*
GA	1.092666667	771.6192479	4.6	0.798382022
K-means	0.024472222	831.5881861	11.5	0.51191103

Berdasarkan Tabel 5.6, dibentuk empat diagram pada Gambar 5.17 - 5.20 untuk menjelaskan perbandingan antara algoritma genetika dengan algoritma *K-means* berdasarkan keempat variabel terikat.



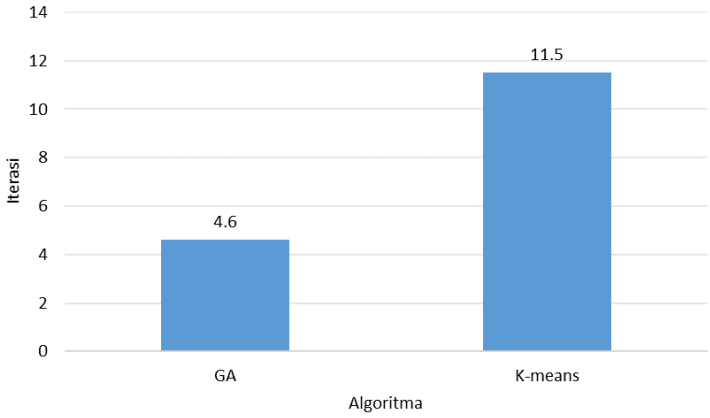
Gambar 5.17: Diagram hubungan algoritma dengan waktu tempuh

Berdasarkan diagram pada Gambar 5.17, 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*.



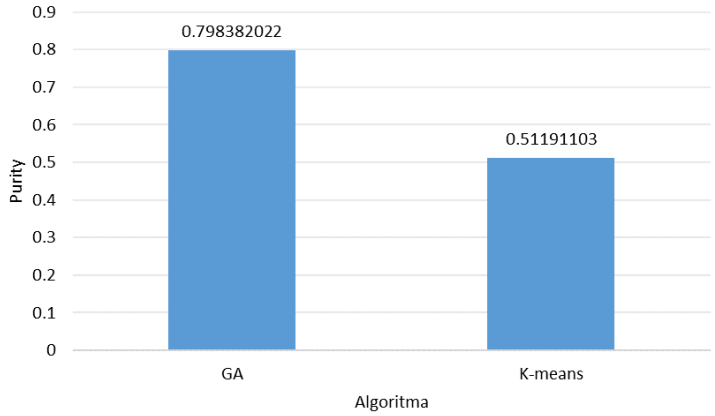
Gambar 5.18: Diagram hubungan algoritma dengan *intracluster similarity*

Berdasarkan diagram pada Gambar 5.18, nilai *intracluster similarity* pada algoritma *K-means* lebih besar 7% daripada algoritma genetika.



Gambar 5.19: Diagram hubungan algoritma dengan banyaknya iterasi

Berdasarkan diagram pada Gambar 5.19, jumlah iterasi dengan menggunakan algoritma *K-means* lebih banyak 150% daripada algoritma genetika. Namun berdasarkan diagram pada Gambar 5.17, 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 5.20: Diagram hubungan algoritma dengan nilai *purity*

Berdasarkan diagram pada Gambar 5.20, nilai *purity* algoritma genetika lebih besar 56% dibandingkan dengan nilai *purity* menggunakan algoritma *K-means*.

5.4 Analisis Hasil Eksperimen

Berikut merupakan hasil analisis berdasarkan hasil eksperimen yang telah dilakukan.

1. *Intracluster similarity* belum dapat menentukan apakah hasil dari suatu pengelompokan sudah baik atau belum. Berdasarkan hasil eksperimen, nilai *intracluster* similarity tidak berbanding lurus dengan nilai *purity*. Salah satu kemungkinannya adalah karena semakin jauh jarak suatu anggota *cluster* dari *centroid*, maka nilai *intracluster similarity* semakin kecil. Nilai *purity* merupakan suatu nilai biner sehingga sejauh apapun suatu anggota *cluster* dari *centroid*, objek tersebut tetap merupakan anggota dari *cluster*. Kemungkinan yang terjadi adalah banyak objek yang jaraknya cukup jauh dari *centroid* namun tetap merupakan bagian dari *cluster* tersebut karena jarak ke *centroid* lain lebih jauh. Hal ini yang dapat menyebabkan suatu hasil pengelompokan memiliki *intracluster similarity* bernilai kecil sedangkan *purity* bernilai besar atau sebaliknya.

2. Algoritma genetika lebih baik 56% menurut nilai *purity* (Gambar 5.20). Namun, kekurangan dari algoritma genetika adalah waktu pemrosesan yang jauh lebih lama dibandingkan dengan algoritma *K-means*.

3. Untuk membandingkan antara algoritma *K-means* dengan algoritma genetika, maka dilakukan operasi statistika dasar terhadap data hasil eksperimen (Tabel B.9 dan Tabel B.1 pada Lampiran B). Hasil perhitungan statistika dasar terhadap kedua tabel tersebut dijelaskan dalam Tabel 5.7.

Tabel 5.7: Hasil perhitungan statistika terhadap hasil eksperimen algoritma genetika dan *K-means*

Algoritma		Rata-rata	Standar Deviasi	Nilai Maksimum	Nilai Minimum
GA	*intracluster*	771.6192479	18.93213605	781.7710303	733.7897158
	purity	0.798382022	0.01228077	0.820224719	0.78741573
K-means	*intracluster*	831.5881861	272.6741901	1383.188573	512.6894755
	purity	0.51191103	0.20861176	0.794606742	0.248153619

Berdasarkan Tabel 5.7, standar deviasi dari nilai *intracluster* pada algoritma *K-means* 1340% lebih besar dibandingkan dengan algoritma genetika. Selain itu, standar deviasi dari nilai

purity pada algoritma *K-means* 1599% lebih besar dibandingkan dengan algoritma genetika. Standar deviasi yang bernilai besar pada algoritma *K-means* membuktikan bahwa pesebaran data cukup luas. Hal ini juga didukung dengan jarak antara nilai maksimum dan minimum yang cukup jauh untuk nilai *intracluster* maupun *purity* pada algoritma *K-means*. Pesebaran data yang meluas membuktikan bahwa hasil dari algoritma *K-means* tidak stabil. Hal ini juga berarti bahwa algoritma *K-means* sering terjebak pada *local optimum*.

Untuk algoritma genetika, standar deviasi untuk kedua variabel tidak begitu besar. Hal ini membuktikan bahwa pesebaran data pada algoritma genetika cukup sempit. Hal ini didukung dengan jarak antara nilai minimum dan maksimum untuk kedua variabel pada algoritma genetika tidak begitu jauh. Pesebaran data yang sempit membuktikan bahwa hasil dari algoritma genetika cukup stabil. Hal ini dapat membuktikan bahwa algoritma genetika lebih baik dalam mengatasi *local optimum* dibandingkan dengan algoritma *K-means*.

BAB 6

KESIMPULAN DAN SARAN

6.1 Kesimpulan

Kesimpulan yang dapat diambil dari penelitian ini adalah sebagai berikut:

1. Berdasarkan *dataset* yang telah digunakan, algoritma genetika dapat digunakan dalam pengelompokan dokumen. Namun, diperlukan beberapa adaptasi terhadap komponen-komponen dalam algoritma genetika sebelum dapat digunakan untuk mengelompokkan dokumen. Adaptasi yang perlu dilakukan di antaranya adalah:

   - Merepresentasikan dokumen ke dalam suatu model ruang vektor.
   - Kromosom tersusun atas K *centroid* dalam bentuk vektor.
   - Fungsi *fitness* yang digunakan adalah menggunakan *intracluster similarity*.

2. Perangkat lunak yang menggunakan algoritma genetika untuk mengelompokkan dokumen telah berhasil dibuat. Berdasarkan hasil eksperimen menggunakan *dataset* dalam penelitian ini, rata-rata nilai *purity* dari hasil pengelompokan menggunakan algoritma genetika adalah sebesar 0.799, lebih baik 56% dibandingkan dengan menggunakan algoritma K-means. 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*.

3. Representasi kromosom yang kurang tepat juga menjadi alasan algoritma genetika berjalan dengan lambat. Mulai generasi kedua, *centroid* yang menyusun kromosom bersifat tidak *sparse* (memiliki sedikit elemen bernilai nol). *Centroid* yang tidak *sparse* memperlambat proses perhitungan menggunakan *cosine similarity* karena seharusnya perhitungan menggunakan *cosine similarity* dapat mengabaikan elemen berbobot nol. Namun karena banyak elemen yang tidak berbobot nol, maka hanya sedikit elemen yang dapat diabaikan dalam proses perhitungan menggunakan *cosine similarity*.

4. Metrik yang digunakan dalam penelitian ini yaitu *intracluster similarity* kurang merepresentasikan seberapa baik suatu hasil pengelompokan. Berdasarkan hasil eksperimen, nilai *intracluster* similarity tidak berbanding lurus dengan nilai *purity*. Salah satu kemungkinannya adalah karena semakin jauh jarak suatu anggota *cluster* dari *centroid*, maka nilai *intracluster similarity* semakin kecil. Nilai *purity* merupakan suatu nilai biner sehingga sejauh apapun suatu anggota *cluster* dari *centroid*, objek tersebut tetap merupakan anggota dari *cluster*. Kemungkinan yang terjadi adalah banyak objek yang jaraknya cukup jauh dari *centroid* namun tetap merupakan bagian dari *cluster* tersebut karena jarak ke *centroid* lain lebih jauh.

6.2 Saran

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

1. Mencari suatu metrik yang lebih mendekati nilai *purity*. Beberapa alternatif metrik yang bisa dicoba untuk mengembangkan penelitian ini adalah dengan menggunakan metrik *intercluster* atau menggunakan metrik *silhouette*.

2. Menggunakan representasi kromosom yang lain sehingga dapat menjaga agar vektor dari *centroid* tetap *sparse*. Hal ini dapat dilakukan dengan mengubah representasi kromosom menjadi dokumen dan keanggotaannya dalam *cluster*.

3. Memproses dokumen dengan tipe selain TXT seperti file dengan ekstensi DOC, DOCX, PDF, dan lain-lain. Selain itu, pengembangan dari penelitian ini adalah dengan memperhitungkan atribut lain dari suatu dokumen selain teks seperti gambar, metadata (penulis dokumen, waktu dibuat) dan lain-lain.

DAFTAR REFERENSI

[1] Gan, G., Ma, C., dan Wu, J. (2007) *Data clustering: theory, algorithms, and applications*. Siam.

[2] 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.

[3] Shah, N. dan Mahajan, S. (2012) Document clustering: a detailed review. *International Journal of Applied Information Systems*, **4**, 30–38.

[4] Maulik, U. dan Bandyopadhyay, S. (2000) Genetic algorithm-based clustering technique. *Pattern recognition*, **33**, 1455–1465.

[5] Holland, J. H. (1992) Genetic algorithms. *Scientific american*, **267**, 66–73.

[6] Sivanandam, S. dan Deepa, S. (2007) *Introduction to Genetic Algorithms*. Springer Science & Business Media.

[7] Zhai, C. dan Massung, S. (2016) *Text data management and analysis: a practical introduction to information retrieval and text mining*. Morgan & Claypool.

[8] Mecca, G., Raunich, S., dan Pappalardo, A. (2007) A new algorithm for clustering search results. *Data & Knowledge Engineering*, **62**, 504–522.

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

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

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

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

[13] 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             String filepath=Params.getInstance().getFilepath();
491             List<Document>[] result=new ArrayList[Params.getInstance().getK()];
492             for (int i = 0; i < result.length; i++) {
493                 result[i]=new ArrayList<>();
494             }
495             for(Entry<Document,Integer> entry:clusteringResult.entrySet()){
496                 result[entry.getValue()].add(entry.getKey());
497             }
498             int maxLength=0;
499             for (int i = 0; i < result.length; i++) {
500                 maxLength=Math.max(maxLength, result[i].size());
501             }
502             DateFormat dateFormat = new SimpleDateFormat("yyyy.MM.dd HH_mm_ss");
503             Date date = new Date();
504             String pathHasil=textFieldHasil.getText();
505             String filename=pathHasil+"\\"+mode+"-"+dateFormat.format(date)+".csv";
506             BufferedWriter bw = new BufferedWriter(new FileWriter(filename));
507 
508             //write header
509             bw.write("Direktori Dokumen:,"+filepath+"\r\n \r\n");
510             bw.write("Parameter: \r\n");
511             if(Params.getInstance().getK()!=-1)bw.write("Banyaknya Cluster,"+Params.getInstance().getK()+"\r\n");
512             if(Params.getInstance().getP()!=-1)bw.write("Banyaknya Populasi,"+Params.getInstance().getP()+"\r\n");
513             if(Params.getInstance().getWeightingMethod()!=-1)bw.write("Metode Pembobotan,"+(Params.getInstance().
                    getWeightingMethod()==0?"TF-IDF":"Frekuensi")+"\r\n");
514             if(Params.getInstance().getMu_m()!=-1)bw.write("Probabilitas Mutasi,"+Params.getInstance().getMu_m()+"\r\n");
515             if(Params.getInstance().getMaxIt()!=-1)bw.write("Maksimum Iterasi,"+Params.getInstance().getMaxIt()+"\r\n");
516             if(Params.getInstance().getElitismCount()!=-1)bw.write("Individu Elitisme,"+Params.getInstance().getElitismCount()+"\r
                    \n");
517             if(Params.getInstance().getConvergeGen()!=-1)bw.write("Banyaknya Generasi Konvergen,"+Params.getInstance().
                    getConvergeGen()+"\r\n");
518             if(Params.getInstance().getConvergeEpsilon()!=-1)bw.write("Batas Konvergen,"+Params.getInstance().getConvergeEpsilon()
                    +"\r\n");
519 
520             //write hasil
521             bw.write("\r\nHasil: \r\n");
522             bw.write("Waktu,"+timeFormatter(runningTime)+"\r\n");
523             bw.write("Intracluster,"+fitness+"\r\n");
524             bw.write("Banyak Iterasi,"+(curIt-1)+"\r\n \r\n");
525 
526             //write hasil clustering
527             bw.write("Hasil Clustering: \r\n");
528             for (int i = 0; i < result.length; i++) {
529                 bw.write("C"+(i+1)+",");
530             }
531             bw.write("\r\n");
532             for (int i = 0; i < maxLength; i++) {
533                 for (int j = 0; j < result.length; j++) {
534                     if(i<result[j].size()){
535                         String name=result[j].get(i).getDocName().substring(filepath.length());
536                         if(name.charAt(0)=='\\'){
537                             name=name.substring(1);
538                         }
539                         bw.write(name);
540                     }
```

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

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 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();
        }
    }

    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(){
        List<Document> docs=GAClusterer.getInstance().getAllDocs();
        HashMap<String, Double> sumOfCentroid[]=new HashMap[genes.size()];
        for (int i = 0; i < genes.size(); i++) {
            sumOfCentroid[i]=new HashMap<>();
        }
        int pointCount[]=new int[genes.size()];

        //assign dokumen ke cluster
        long cur=System.currentTimeMillis();
        determineCluster(docs);
        cur=System.currentTimeMillis();

        //hitung centroid baru
        for (int i = 0; i < docs.size(); i++) {
            int clusterCode=clusteringResult.get(docs.get(i));
            for(String term:docs.get(i).getVector().getKeySet()){
                double curValue=sumOfCentroid[clusterCode].containsKey(term)?sumOfCentroid[clusterCode].get(term):0;
                curValue+=docs.get(i).getVector().getWeight(term);
                if(curValue!=0){
                    sumOfCentroid[clusterCode].put(term, curValue);
                }
            }
            pointCount[clusterCode]++;
```

```
100         }
101
102         cur=System.currentTimeMillis();
103
104         for (int i = 0; i < genes.size(); i++) {
105             for(String term:Lexicon.getInstance().getAllTermList()){
106                 double nextValue=sumOfCentroid[i].containsKey(term)?sumOfCentroid[i].get(term):0;
107                 nextValue=pointCount[i]==0.0?0.0:nextValue/pointCount[i];
108                 if(nextValue!=0){
109                     genes.get(i).getValue().setWeight(term, nextValue);
110                 }
111             }
112         }
113         this.fitnessValue=-1;
114         return getFitness();
115     }
116
117     public double getFitness(){
118         if(fitnessValue!=-1)return this.fitnessValue;
119         List<Document> docs=GAClusterer.getInstance().getAllDocs();
120         double res=0.0;
121         determineCluster(docs);
122         for (int i = 0; i < docs.size(); i++) {
123             int clusterCode=clusteringResult.get(docs.get(i));
124             double tmp=docs.get(i).getVector().calculateSimilarity(this.genes.get(clusterCode).getValue());
125             res+=tmp;
126         }
127         fitnessValue=res;
128         return res;
129     }
130
131     public List<Gene> getAllGenes() {
132         return genes;
133     }
134
135     public HashMap<Document, Integer> getClusteringResult() {
136         return clusteringResult;
137     }
138
139     @Override
140     public int compareTo(Chromosome o) {
141         return this.getFitness()>o.getFitness()?-1:this.getFitness()<o.getFitness()?1:0;
142     }
143 }
```

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()) {
89                      q.poll();
90                      q.offer(cur);
91                  }
92              }
93          }
94          ArrayList<Chromosome> res=new ArrayList<>(q);
95          return res;
96      }
97
98      public List<Chromosome> selection(int elitismCount) {
99          long cur = System.currentTimeMillis();
100         List<Chromosome> nextGen = elitism(elitismCount);
101         cur = System.currentTimeMillis();
102         for (int i = 0; i < population.size() - elitismCount; i++) {
103             Chromosome parentA = this.rouletteWheelSelect();
104             Chromosome parentB = this.rouletteWheelSelect();
105             Chromosome offspring = parentA.crossover(parentB);
106             offspring.mutate();
107             nextGen.add(offspring);
108             progress.set(0.6+(0.2*i/population.size()));
109         }
110         return nextGen;
111     }
112
113     public void initialize() {
114         String filepath = Params.getInstance().getFilepath();
115         int k = Params.getInstance().getK();
116         int popSize = Params.getInstance().getP();
117
118         long cur = System.currentTimeMillis();
119         //document indexing
120         File folder = new File(filepath);
121         File[] files = folder.listFiles();
122         Queue<File> queue = new LinkedList<>();
123         for (File file : files) {
124             queue.offer(file);
125         }
126         LinkedList<File> allFiles = new LinkedList<>();
127
128         while (!queue.isEmpty()) {
129             File tmp = queue.poll();
130             if (tmp.isDirectory()) {
131                 File[] tmpFolder = tmp.listFiles();
132                 for (int i = 0; i < tmpFolder.length; i++) {
133                     queue.offer(tmpFolder[i]);
134                 }
135             } else {
136                 allFiles.add(tmp);
137             }
138         }
139
140         Lexicon.getInstance().setNumberOfDocument(allFiles.size());
141
142         allFiles.forEach((file) -> {
143             docs.add(new Document(file));
144         });
145
146         List<Document> list = new LinkedList<>();
147         list.addAll(docs);
148
149         cur = System.currentTimeMillis();
150         //generate initial population
```

```
151         for (int i = 0; i < popSize; i++) {
152             Chromosome temp = new Chromosome();
153             Collections.shuffle(list);
154             for (int j = 0; j < k; j++) {
155                 temp.addGene(new Gene(list.get(j).getVector()));
156             }
157             population.add(temp);
158         }
159 
160     }
161 
162     public List<Document> getAllDocs() {
163         return docs;
164     }
165 
166     public void cluster() {
167         for (int a = 0; a < Params.getInstance().getMaxIt(); a++) {
168             progress.set(0);
169             if(!isRunning){
170                 progress.set(0);
171                 return;
172             }
173             int popSize = Params.getInstance().getP();
174             int elitismCount = Params.getInstance().getElitismCount();
175             int convergeGen = Params.getInstance().getConvergeGen();
176             double convergeEpsilon = Params.getInstance().getConvergeEpsilon();
177             long cur = System.currentTimeMillis();
178 
179             //compute fitness : 0.6
180             double totalTime = 0;
181             for (int i = 0; i < popSize; i++) {
182                 if(!isRunning){
183                     progress.set(0);
184                     return;
185                 }
186                 Chromosome c=population.get(i);
187                 cur = System.currentTimeMillis();
188                 c.computeFitness();
189                 progress.set(0.6*(i*1.0/popSize));
190                 totalTime += (System.currentTimeMillis() - cur) / 1000.0;
191             }
192             cur = System.currentTimeMillis();
193 
194             if(!isRunning){
195                 progress.set(0);
196                 return;
197             }
198             //selection
199             population = selection(elitismCount);
200             if(!isRunning){
201                 progress.set(0);
202                 return;
203             }
204             Chromosome solution = null;
205             for (int j = 0; j < popSize; j++) {
206                 Chromosome temp = population.get(j);
207                 if (solution == null || solution.getFitness() < temp.getFitness()) {
208                     solution = temp;
209                 }
210                 progress.set(0.8+(0.2*j/popSize));
211             }
212             if(!isRunning){
213                 progress.set(0);
214                 return;
215             }
216 
217             this.solutionList.add(solution);
218 
219             if (this.solutionList.size() > convergeGen) {
220                 double delta = 0;
221                 this.solutionList.remove(0);
222                 for (int j = 1; j < this.solutionList.size(); j++) {
223                     double curr = this.solutionList.get(j).getFitness();
224                     double prev = this.solutionList.get(j - 1).getFitness();
225                     delta += Math.abs(curr - prev);
226                 }
227                 delta /= this.solutionList.size();
228                 if (delta < convergeEpsilon) {
229                     a=Params.getInstance().getMaxIt();
230                 }
231             }
232             progress.set(1);
233             if(!isRunning){
234                 progress.set(0);
235                 return;
236             }
237         }
238     }
239 
240     public Chromosome getSolution(){
241         return this.solutionList.get(this.solutionList.size()-1);
242     }
243 
244     public void reset(){
245         instance=null;
246     }
247 }
```

Listing A.6: Gene.java

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


import ga.clustering.gui.IR.Vector;
import java.util.HashMap;

/*
 * 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) {
        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;

```

```java
/**
 *
 * @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();
        }
        while(sc.hasNext())
        {
            String temp=sc.next();
            temp=temp.replaceAll("[^A-Za-z0-9]", "").toLowerCase();
            if(temp.length()==0)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

```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

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

```

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

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

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

    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 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){
```

```
56          if(!termsWeight.containsKey(term))return 0.0;
57          return termsWeight.get(term);
58      }
59
60      public double calculateWeight(String term, HashMap<String, Integer> wordCount){
61          return termWeighting.calculateWeight(term, wordCount);
62      }
63
64      public double calculateSimilarity(Vector otherVector) {
65          return similarityCalculator.calculateSimilarity(this, otherVector);
66      }
67
68      public void mutate(){
69          Random rand=new Random();
70          String key=(String) termsWeight.keySet().toArray()[rand.nextInt(termsWeight.size())];
71          double value=termsWeight.get(key);
72          double newVal=value+(rand.nextBoolean()?value*2*rand.nextDouble():value*-2*rand.nextDouble());
73          newVal=newVal<0?0:newVal;
74          termsWeight.put(key, newVal);
75      }
76
77      public void setTermsWeight(HashMap<String, Double> termsWeight) {
78          this.termsWeight = new HashMap<>(termsWeight);
79      }
80
81      public void setWeight(String term, double value){
82          termsWeight.put(term,value);
83      }
84
85      public double getLength(){
86          double res=0.0;
87          for(Map.Entry<String,Double> entry: termsWeight.entrySet()){
88              res+=(entry.getValue()*entry.getValue());
89          }
90          res=Math.sqrt(res);
91          return res;
92      }
93
94      public Set<String> getKeySet(){
95          return termsWeight.keySet();
96      }
97
98      public HashMap<String, Double> getTermsWeight() {
99          return termsWeight;
100     }
101
102     public int getDimension(){
103         return this.termsWeight.size();
104     }
105 }
```

Listing A.15: KMeans.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.KMeans;
7
8
9  import ga.clustering.gui.Params;
10 import ga.clustering.gui.IR.Document;
11 import ga.clustering.gui.IR.Lexicon;
12 import ga.clustering.gui.IR.Vector;
13 import java.io.File;
14 import java.util.Collections;
15 import java.util.HashMap;
16 import java.util.LinkedList;
17 import java.util.List;
18 import java.util.Queue;
19 import javafx.beans.property.ReadOnlyDoubleProperty;
20 import javafx.beans.property.ReadOnlyDoubleWrapper;
21
22 /**
23  *
24  * @author Cornelius David
25  */
26 public class KMeans {
27
28     private List<Document> docs;
29     private HashMap<Document, Integer> solution;
30     private static KMeans instance;
31     private double solutionIntracluster;
32
33     private boolean isRunning;
34
35     private final ReadOnlyDoubleWrapper progress = new ReadOnlyDoubleWrapper();
36
37     private KMeans(){
38         docs=new LinkedList<>();
39         solutionIntracluster=-1;
40         this.isRunning=true;
41     }
42
43     public void setIsRunning(boolean isRunning) {
44         this.isRunning = isRunning;
45     }
```

```
46
47      public double getProgress() {
48          return progressProperty().get();
49      }
50
51      public ReadOnlyDoubleProperty progressProperty() {
52          return progress ;
53      }
54
55      public HashMap<Document, Integer> getSolution() {
56          return solution;
57      }
58
59      public static KMeans getInstance(){
60          if(instance==null)instance=new KMeans();
61          return instance;
62      }
63
64      public void cluster() {
65          int k=Params.getInstance().getK();
66          String filepath=Params.getInstance().getFilepath();
67          int maxIt=Params.getInstance().getMaxIt();
68          File folder = new File(filepath);
69          File[] files = folder.listFiles();
70          Queue<File> queue = new LinkedList<>();
71          for (int i = 0; i < files.length; i++) {
72              queue.offer(files[i]);
73          }
74          LinkedList<File> allFiles = new LinkedList<>();
75
76          while (!queue.isEmpty()) {
77              File tmp = queue.poll();
78              if (tmp.isDirectory()) {
79                  File[] tmpFolder = tmp.listFiles();
80                  for (int i = 0; i < tmpFolder.length; i++) {
81                      queue.offer(tmpFolder[i]);
82                  }
83              } else {
84                  allFiles.add(tmp);
85              }
86          }
87
88          Lexicon.getInstance().setNumberOfDocument(allFiles.size());
89
90          for (File file : allFiles) {
91              docs.add(new Document(file));
92          }
93
94          //init cluster
95          List<Document> list = new LinkedList<>();
96          list.addAll(docs);
97          Collections.shuffle(list);
98          Vector[] centroids = new Vector[k];
99          for (int i = 0; i < centroids.length; i++) {
100             centroids[i] = list.get(i).getVector();
101         }
102
103         if(!isRunning){
104             progress.set(0);
105             return;
106         }
107
108         iteration:
109         for (int a = 0; a < maxIt; a++) {
110             HashMap<Document, Integer> cluster=determineCluster(docs, centroids);
111             progress.set(0);
112
113             if(!isRunning){
114                 progress.set(0);
115                 return;
116             }
117
118             HashMap<String, Double> sumOfCentroid[] = new HashMap[k];
119             for (int i = 0; i < k; i++) {
120                 sumOfCentroid[i]=new HashMap<>();
121             }
122             int pointCount[] = new int[k];
123             for (int i = 0; i < docs.size(); i++) {
124                 int clusterCode = cluster.get(docs.get(i));
125                 for (String term : docs.get(i).getVector().getKeySet()) {
126                     double curValue = sumOfCentroid[clusterCode].containsKey(term) ? sumOfCentroid[clusterCode].get(term) : 0;
127                     curValue += docs.get(i).getVector().getWeight(term);
128                     if (curValue != 0) {
129                         sumOfCentroid[clusterCode].put(term, curValue);
130                     }
131                 }
132                 if(!isRunning){
133                     progress.set(0);
134                     return;
135                 }
136                 pointCount[clusterCode]++;
137                 progress.set(i*0.5/docs.size());
138             }
139             for (int i = 0; i < k; i++) {
140                 HashMap<String,Double> temp=new HashMap<>();
141                 for (String term : Lexicon.getInstance().getAllTermList()) {
142                     double nextValue = sumOfCentroid[i].containsKey(term) ? sumOfCentroid[i].get(term) : 0;
143                     if(!isRunning){
144                         progress.set(0);
```

```
145                         return;
146                     }
147                     nextValue = pointCount[i] == 0.0 ? 0.0 : nextValue / pointCount[i];
148                     if (nextValue != 0) {
149 //                        centroids[i].setWeight(term, nextValue);
150                         temp.put(term, nextValue);
151                     }
152                 }
153                 centroids[i].setTermsWeight(temp);
154                 progress.set(0.5+(i*0.49)/k);
155             }
156 
157             if(!isRunning){
158                 progress.set(0);
159                 return;
160             }
161             progress.set(1);
162             if(solution==null)solution=cluster;
163             else{
164                 for (int i = 0; i < docs.size(); i++) {
165                     int before=solution.get(docs.get(i));
166                     int cur=cluster.get(docs.get(i));
167                     if(before!=cur){
168                         solution=cluster;
169                         continue iteration;
170                     }
171                 }
172                 solutionIntracluster=computeIntracluster(cluster, centroids);
173                 return;
174             }
175         }
176         solutionIntracluster=computeIntracluster(solution, centroids);
177     }
178 
179     public double getSolutionIntracluster() {
180         return solutionIntracluster;
181     }
182 
183     private double computeIntracluster(HashMap<Document, Integer> cluster, Vector[] centroids){
184         double res=0.0;
185         for (int i = 0; i < docs.size(); i++) {
186             int clusterCode=cluster.get(docs.get(i));
187             double tmp=docs.get(i).getVector().calculateSimilarity(centroids[clusterCode]);
188             res+=tmp;
189         }
190         return res;
191     }
192 
193     private HashMap<Document, Integer> determineCluster(List<Document> docs, Vector[] centroids){
194         HashMap<Document, Integer> clusteringResult=new HashMap<>();
195         for (int i = 0; i < docs.size(); i++) {
196             int cluster=-1;
197             double similarity=Double.MIN_VALUE;
198             for (int j = 0; j < centroids.length; j++) {
199                 double temp=docs.get(i).getVector().calculateSimilarity(centroids[j]);
200                 if(temp>similarity){
201                     similarity=temp;
202                     cluster=j;
203                 }
204             }
205             clusteringResult.put(docs.get(i), cluster);
206         }
207         return clusteringResult;
208     }
209 
210     public void reset(){
211         instance=null;
212     }
213 }
```

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.

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

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

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.

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

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

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.

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

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

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.

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

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

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.

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

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

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.

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

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

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

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

Tabel B.7: Hasil eksperimen kasus uji 7 (Individu elitisme=0)

Waktu	Intercluster	Iterasi	Purity
5 jam 55 menit 40 detik	775.2651111	15	0.789213483
6 jam 3 menit 59 detik	773.4957837	15	0.800898876
6 jam 18 menit 24 detik	775.5719472	15	0.795505618
5 jam 42 menit 27 detik	540.8704844	15	0.977078652
5 jam 48 menit 31 detik	533.7854398	15	0.768539326

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 berdasarkan parameter diatas dijelaskan dalam Tabel B.8.

Tabel B.8: Hasil eksperimen kasus uji 8 (Individu elitisme=5)

Waktu	Intercluster	Iterasi	Purity
2 jam 5 menit 57 detik	1206.637007	6	0.379775281
2 jam 16 menit 44 detik	1725.804155	6	0.315955056
2 jam 32 menit 16 detik	1676.015285	7	0.360898876
2 jam 3 menit 35 detik	545.358012	6	0.966292135
1 jam 26 menit 0 detik	922.8631224	5	0.61258427

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.9.

Tabel B.9: Hasil eksperimen algoritma K-means

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

LAMPIRAN C

CONTOH DATASET

Berikut merupakan beberapa contoh dokumen dalam dataset BBC (3.1) dengan mengambil satu buah dokumen untuk setiap topiknya.

C.1 Topik *Business*

Listing C.1: business.txt

```
Ad sales boost Time Warner profit

Quarterly profits at US media giant TimeWarner jumped 76% to $1.13bn (\pounds 600m) for the three months to December, from $639m
      year-earlier.

The firm, which is now one of the biggest investors in Google, benefited from sales of high-speed internet connections and higher
      advert sales. TimeWarner said fourth quarter sales rose 2% to $11.1bn from $10.9bn. Its profits were buoyed by one-off gains
       which offset a profit dip at Warner Bros, and less users for AOL.

Time Warner said on Friday that it now owns 8% of search-engine Google. But its own internet business, AOL, had has mixed fortunes
      . It lost 464,000 subscribers in the fourth quarter profits were lower than in the preceding three quarters. However, the
      company said AOL's underlying profit before exceptional items rose 8% on the back of stronger internet advertising revenues.
       It hopes to increase subscribers by offering the online service free to TimeWarner internet customers and will try to sign
      up AOL's existing customers for high-speed broadband. TimeWarner also has to restate 2000 and 2003 results following a probe
       by the US Securities Exchange Commission (SEC), which is close to concluding.

Time Warner's fourth quarter profits were slightly better than analysts' expectations. But its film division saw profits slump 27%
       to $284m, helped by box-office flops Alexander and Catwoman, a sharp contrast to year-earlier, when the third and final
      film in the Lord of the Rings trilogy boosted results. For the full-year, TimeWarner posted a profit of $3.36bn, up 27% from
       its 2003 performance, while revenues grew 6.4% to $42.09bn. "Our financial performance was strong, meeting or exceeding all
       of our full-year objectives and greatly enhancing our flexibility," chairman and chief executive Richard Parsons said. For
      2005, TimeWarner is projecting operating earnings growth of around 5%, and also expects higher revenue and wider profit
      margins.

TimeWarner is to restate its accounts as part of efforts to resolve an inquiry into AOL by US market regulators. It has already
      offered to pay $300m to settle charges, in a deal that is under review by the SEC. The company said it was unable to
      estimate the amount it needed to set aside for legal reserves, which it previously set at $500m. It intends to adjust the
      way it accounts for a deal with German music publisher Bertelsmann's purchase of a stake in AOL Europe, which it had
      reported as advertising revenue. It will now book the sale of its stake in AOL Europe as a loss on the value of that stake.
```