# BAB II
# TINJAUAN PUSTAKA

## 2.1 Restoran

Restoran merupakan suatu tempat yang menyediakan makanan dan minuman untuk dikonsumsi oleh tamu, sebagai kebutuhan yang sangat mendasar akan makan dan minum dalam rangka memulihkan kembali kondisinya yang telah berkurang setelah melakukan suatu kegiatan sehingga bisa kembali pada stamina yang semula. Restoran ada yang berlokasi dalam suatu hotel, kantor maupun pabrik, dan banyak juga yang berdiri sendiri di luar bangunan itu. Semua jenis restoran pada umunya mempunyai suatu kepentingan dan tujuan yang sama, yaitu menyediakan dan menyajikan makanan dan minuman kepada umum dengan tujuan untuk memperoleh keuntungan sesuai dengan yang diinginkan. Berdasarkan cara pengelolaan manajemen dan operasionalnya, restoran dapat dibedakan sebagai berikut (Wiwoho, 2008):

a. Restoran yang dikelola dengan manajemen sendiri yang tidak ada kaitannya dengan hotel (*self operation*).
b. Restoran yang dikelola oleh manajemen hotel, yakni restoran sebagai salah satu fasilitas hotel (*integrated to the hotel*).

Pada umumnya, restoran dalam menyajikan menu sajiannya berada di tempat secara langsung. Namun, dalam perkembangannya sekarang ini ada juga yang menyediakan layanan *take out* atau *delivery service* sebagai salah satu bentuk pelayanan kepada konsumennya. Klasifikasi jenis-jenis restoran dari umum, terbagi menjadi 3 jenis (Sumarsono, 2015):

*a. Formal Dining Room*

Restoran ini merupakan jenis restoran yang merupakan restoran kelas atas. Restoran ini memiliki format yang eksklusif yang diperuntukkan tamu tertentu yang dapat menikmatinya. Jenis restoran ini biasanya terdapat di hotel, sebagai tempat yang tidak lebih ekonomis daripada tempat makan biasa. Jika merujuk pada keberadaannya di hotel, dining room pada dasarnya disediakan untuk para tamu yang tinggal di hotel tersebut, namun juga bersifat terbuka bagi tamu dari luar.

*b. Informal Dining Room*

Restoran ini merupakan jenis restoran yang tidak formal. Penjabaran restoran informal merupakan industri jasa pelayanan makanan dan minuman yang dikelola secara komersial dan profesional dengan lebih mengutamakan kecepatan pelayanan, kepraktisan, dan percepatan frekuensi yang silih berganti pelanggan.

c. *Specialities Restaurant*

Restoran jenis ini merupakan industri jasa pelayanan makanan dan minuman yang dikelola secara komersial dan profesional dengan menyediakan makanan khas dan diikuti dengan sistem penyajian yang khas dari suatu negara tersebut, seperti *Indonesian food restaurant*, *Chinese food restaurant*, *Japanese food restaurant*, dan lain-lain yang serupa sesuai kekhasan negara atau daerah tertentu.

## 2.2 Sistem Rekomendasi

Sistem rekomendasi merupakan aplikasi perangkat lunak yang menyediakan informasi *item* yang diperkirakan bernilai untuk tugas rekayasa perangkat lunak dalam konteks tertentu. Sistem rekomendasi sangat penting dalam lingkungan sosial, pengguna berbagi akses ke serangkaian sumber daya yang sama. Keragaman karakteristik pengguna yang penting, seperti latar belakang mereka, minat khusus mereka, tingkat keahlian mereka menimbulkan masalah menarik dalam hal mengusulkan sumber daya yang menarik, berguna dan dapat dipahami oleh pengguna tertentu (Manouselis et al., 2011).

Ide dasar sistem rekomendasi adalah untuk memanfaatkan berbagai sumber data ini untuk menarik minat pelanggan. Entitas rekomendasi tersebut diberikan disebut sebagai pengguna dan produk yang direkomendasikan disebut sebagai *item*. Oleh karena itu, analisis rekomendasi sering didasarkan pada interaksi sebelumnya antara pengguna dan *item*, karena minat dan kecenderungan di masa lalu merupakan indikator yang baik untuk pilihan di masa depan. Prinsip dasar rekomendasi adalah bahwa ada ketergantungan signifikan antara pengguna dan aktivitas *item*-sentris. Dalam banyak kasus, berbagai kategori *item* dapat ditampilkan korelasi yang signifikan, yang dapat dimanfaatkan untuk membuat rekomendasi yang lebih akurat. Dependensi dapat hadir pada *granularity* yang lebih halus dari masing-masing *item* daripada kategori. Ketergantungan ini dapat dipelajari dengan cara yang didorong oleh data dari matriks penilaian dan model yang dihasilkan digunakan untuk membuat prediksi bagi pengguna target. Semakin besar jumlah *item* yang dinilai yang tersedia untuk pengguna, semakin mudah untuk membuat prediksi yang kuat tentang perilaku pengguna di masa depan (Aggarwal, 2016).

Dalam praktiknya, sistem rekomendasi bisa lebih kompleks dan kaya akan data dengan beragam tipe data tambahan. Model dasar sistem rekomendasi bekerja dengan dua jenis data, yaitu interaksi antara *item* dan pengguna (seperti penilaian atau perilaku pembelian) dan informasi atribut tentang pengguna dan *item* (seperti profil teks atau kata kunci yang relevan). Metode yang menggunakan interaksi antara *item* dan pengguna disebut sebagai metode *collaborative filtering*, sedangkan metode yang menggunakan informasi atribut tentang

pengguna dan item disebut sebagai metode *content-based recommender*. *Content-based recommender* juga menggunakan matriks penilaian dalam banyak kasus. Dalam sistem rekomendasi *knowledge-based*, rekomendasi didasarkan pada persyaratan pengguna yang ditentukan secara eksplisit. Alih-alih menggunakan peringkat historis atau membeli data, basis pengetahuan eksternal dan kendala digunakan untuk membuat rekomendasi. Beberapa sistem rekomendasi menggabungkan berbagai aspek yang berbeda untuk menciptakan sistem gabungan. Sistem gabungan dapat menggabungkan kekuatan berbagai jenis sistem rekomendasi untuk menciptakan teknik yang dapat bekerja lebih kuat dalam berbagai pengaturan (Aggarwal, 2016)

### 2.2.1 Collaborative Filtering Recommender System

*Collaborative filtering* menggunakan kekuatan kolaboratif dari *rating* yang diberikan oleh banyak pengguna untuk membuat rekomendasi. Kebanyakan model *collaborative filtering* berfokus pada hubungan antar *item* atau hubungan antar pengguna untuk proses prediksi (Aggarwal, 2016).

*Collaborative filtering* bergantung pada dua jenis data, yaitu sekumpulan pengguna dan sekumpulan *item*. Hubungan antara pengguna dan *item* terutama dinyatakan dalam *rating* yang disediakan oleh pengguna dan dieksploitasi dalam sesi rekomendasi di masa mendatang untuk memprediksi peringkat yang akan disediakan pengguna untuk *item* tertentu. Langkah pertama dari sistem rekomendasi adalah untuk mengidentifikasi tetangga terdekat (pengguna dengan perilaku peringkat yang sama dibandingkan dengan pengguna saat ini) dan untuk mengekstrapolasi dari *rating* pengguna yang sama dengan *rating* pengguna saat ini (Robillard et al., 2010)

Terdapat dua metode yang umunya digunakan dalam *collaborative filtering*, yaitu (Aggarwal, 2016):

*a.* Metode *memory-based*

Metode *memory-based* juga disebut sebagai algoritma *collaborative filtering neighborhood-based*. Metode ini diprediksi berdasarkan lingkungan sehingga diperoleh peringkat kombinasi *item* dengan pengguna. Lingkungan ini dapat didefinisikan dalam dua cara, yaitu *user-based collaborative filtering* dan *item-based collaborative filtering*. Kedua varian memprediksi sejauh mana pengguna aktif akan tertarik pada *item* yang belum dinilai olehnya hingga saat ini (Robillard et al., 2010).

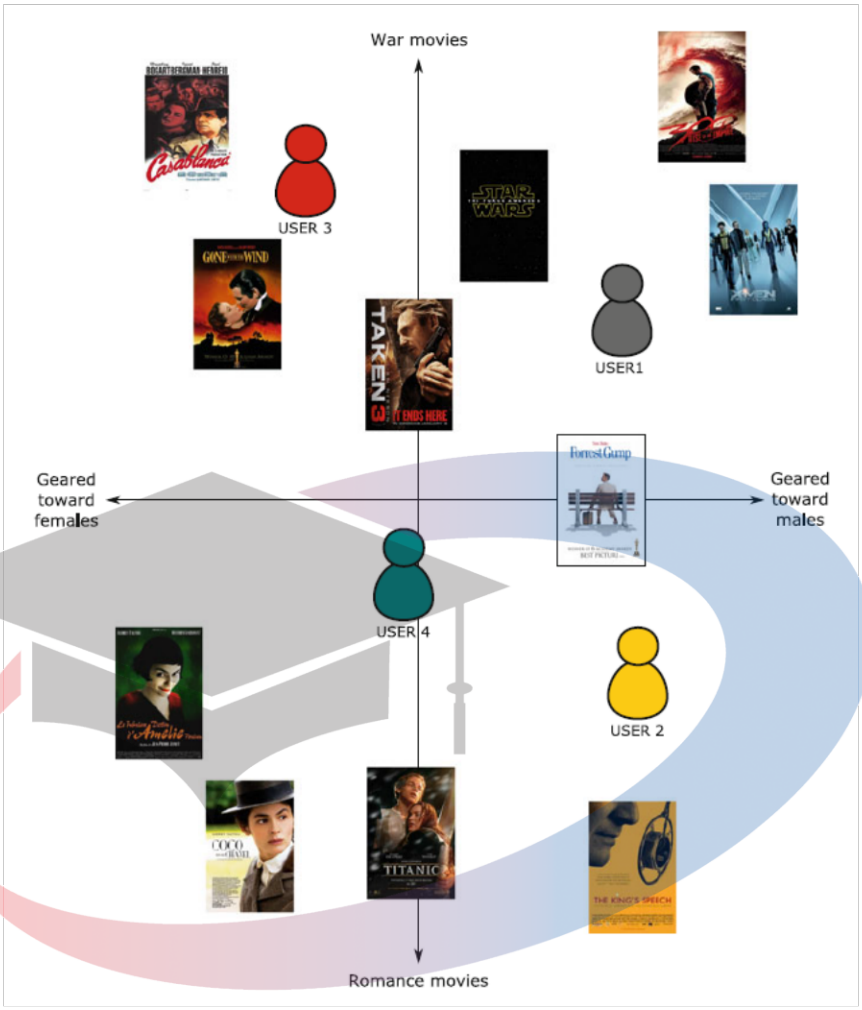
Keuntungan dari metode ini adalah mudah diterapkan dan rekomendasi yang dihasilkan sebagian besar mudah dijelaskan. Di sisi lain, metode ini tidak bekerja dengan baik dengan matriks peringkat yang jauh. Metode ini tidak memiliki cakupan prediksi-prediksi yang luas.

*b.* Metode *model-based*

Pada metode *model-based*, *machine learning* dan metode *data mining* digunakan dalam konteks model prediksi. Model ini diparameterisasi dan parameter model ini dipelajari dalam konteks kerangka kerja optimasi. Metode *model-based* mengembangkan rekomendasi dengan menggunakan model *parametric* atau *semi-parametric* dan *training* pada data *rating* (Bhatnagar, 2016). Metode *model-based* mengelompokkan pengguna yang berbeda dalam *database* ke dalam beberapa *class* berdasarkan pola dari *rating* pengguna (Gong et al., 2009). Keunggulan metode *model-based* dibandingkan dengan metode *neighborhood-based* adalah efisiensi dari segi ruang, kecepatan prediksi dan mencegah *overfitting* (Aggarwal, 2016).

### 2.2.2 Matrix Factorization

*Matrix factorization* adalah proses memfaktorkan matriks menjadi produk matriks. *Matrix factorization* mengeksploitasi asosiasi laten yang ada pada data, misalnya antara pengguna dengan *item*. Secara sederhana, *matrix factorization* menggunakan dua matriks, untuk memetakan informasi korelasi antara faktor pengguna-fitur dan faktor *item*-fitur (Symeonidis & Zioupos, 2016).




Gambar 2.1 Ilustrasi faktor laten, yang mengkategorikan pengguna dan film menggunakan dua axis(pria dengan wanita dan peperangan dengan romantis)

Sumber: (Symeonidis & Zioupos, 2016)

Salah satu kelebihan *matrix factorization* adalah memungkinkan penambahan informasi tambahan. Ketika umpan balik eksplisit tidak tersedia, sistem rekomendasi dapat mengambil preferensi pengguna berdasarkan umpan balik implisit. Umpan balik implisit merefleksikan opini dengan mengamati kebiasaan pengguna seperti sejarah belanja, sejarah *browsing*, pola pencarian bahkan sampai pergerakan mouse (Symeonidis & Zioupos, 2016).

### 2.2.3 Singular Value Decomposition (SVD)

SVD berdasar pada teorema dari aljabar linear yang menyatakan bahwa matriks persegi A dapat difaktorkan menjadi tiga matriks, yaitu matriks ortogonal U, matriks diagonal S, dan transpos dari matriks ortogonal V. Teorema ini umumnya dijabarkan dengan persamaan (1).

$$A_{mn} = U_{mr}\, S_{rr}\, V_{nr}^{T} \quad (1)$$

Pada persamaan tersebut $U^T U = I$ dan $V^T V = I$; I adalah matriks identitas. Kolom dari $U$ adalah vektor eigen ortonormal dari $A\,A^T$, kolom dari V adalah vektor eigen ortonomal dari $A^T A$, dan $S$ adalah matriks diagonal yang menyimpan akar kuadrat dari nilai eigen dari $U$ dan $V$ yang diurutkan menurun (Baker, 2013).

### 2.2.4 Pengujian Keakuratan Hasil Rekomendasi

Untuk menilai metode pengelompokan yang diusulkan, digunakan metode pengukuran *Mean Absolute Error* (MAE). MAE adalah pengukuran rata-rata *error* yang lebih netral, dan tidak ambigu dalam mengukur rata-rata *error* (Willmott & Matsuura, 2005). MAE sama dengan *rating* ($p_{u.i}$) dengan sistem melakukan prediksi terhadap pengguna yang aktif u terhadap *item* i dan kemudian di kurang dengan *rating* ($r_{u.i}$) yang diberikan oleh pengguna u terhadap *item* i. Perhitungan MAE terhadap pengguna aktif u dapat dilihat pada persamaan (2).

$$MAE_u = \frac{1}{n_u} \sum_{i=1}^{n_u} |p_{u.i} - r_{u.i}| \quad (2)$$

$n_u$ adalah kardinalitas dari pengujian rating yang ditentukan untuk pengguna aktif u. Rumus untuk melakukan perhitungan jumlah MAE dapat dilihat pada persamaan (3).

$$MAE = \frac{1}{\#U} \sum_{u=1}^{\#U} MAE_u \quad (3)$$

Dengan #U adalah banyaknya jumlah pengguna yang sedang aktif dalam sistem (Mohammadpour et al., 2019)

## 2.3 Local Search Algorithm

*Local search algorithm* adalah algoritma iteratif yang memulai pencariannya dari sebuah titik dan kemudian ditarik secara acak. Mekanisme generasi kemudian diterapkan secara berturut-turut untuk menemukan solusi yang lebih baik (dalam hal ini fungsi tujuan nilai), dengan menjelajahi lingkungan solusi saat ini. Jika solusi seperti itu ditemukan, itu akan menjadi solusi saat ini. Algoritma berakhir ketika tidak ada peningkatan yang dapat ditemukan dan solusi saat ini dianggap sebagai solusi yang telah di optimasi. Untuk menghindari terperangkap dalam *local* minimum, maka perlu untuk menentukan proses yang menerima transisi keadaan yang mengurangi kinerja dari solusi saat ini. Ini adalah prinsip utama dari *simulated annealing* (SA) (Nikolaev & Jacobson, 2010).

*Local search algorithm* beroperasi menggunakan sebuah *current node* tunggal dan umumnya hanya berpindah ke *node* tetangga dari *node* tersebut. Biasanya jalur yang diikuti oleh pencarian tidak dipertahankan. Meskipun *local search algorithm* tidak sistematis, tetapi *local search algorithm* memiliki dua keunggulan utama, yaitu tempat penyimpanan yang

digunakan sangat kecil atau biasanya dalam jumlah yang konstan dan didapatkan solusi yang masuk akal dalam cakupan yang besar atau tak terbatas. Tetapi keunggulan-keunggulan tersebut tidak cocok dengan algoritma yang bekerja secara sistematis. Selain untuk mendapatkan tujuan, *local search algorithm* juga sangat berguna dalam menyelesaikan masalah-masalah optimisasi, tujuannya adalah untuk menemukan solusi terbaik sesuai dengan tujuan yang diinginkan. Sebuah *local search algorithm* yang sempurna akan selalu mendapatkan solusi (Russell & Norvig, 2010).

### 2.3.1 Genetic Algorithm

*Genetic Algorithm* (GA) adalah suatu algoritma pencarian yang meniru mekanisme dari genetika alam. Kelangsungan dari *fittest* digabungkan di antara struktur *string* dengan pertukaran informasi terstruktur namun acak untuk membentuk sebuah algoritma pencarian. GA secara teoritis dan empiris terbukti untuk menyediakan pencarian yang kuat di ruang yang kompleks. Terdapat 4 perbedaan antara GA dengan prosedur optimisasi normal dan pencarian lainnya, yaitu (Goldberg, 1989):

a. GA bekerja dengan pengkodean dari *set* parameter, bukan parameter itu sendiri.
b. Pencarian dengan GA dimulai dari sebuah populasi, bukan dari *single point*.
c. GA menggunakan informasi hasil yang objektif, bukan turunan atau pengetahuan tambahan lainnya.
d. GA menggunakan aturan transisi probabilistik, bukan aturan deterministik. GA menggunakan aturan transisi probabilistik untuk memandu pencarian mereka dan menggunakan pilihan acak sebagai alat untuk memandu pencarian menuju wilayah ruang pencarian yang memiliki kemungkinan peningkatan.

GA bekerja dari suatu populasi. Dengan cara ini, GA menemukan keamanan dalam angka. Dengan mempertahankan populasi titik sampel yang disesuaikan dengan baik, kemungkinan mencapai puncak yang salah akan berkurang. GA memproses kesamaan dalam pengkodean yang mendasarinya bersama dengan informasi peringkat dari struktur berdasarkan kemampuan bertahan dalam sebuah lingkungan. Dengan mengeksploitasi informasi yang tersedia secara luas, GA dapat diterapkan di hampir semua masalah (Goldberg, 1989)

Pertama, menggunakan koefisien korelasi Pearson, tingkat kesamaan antara *item* dalam sistem dihitung dengan menggunakan persamaan (4).

$$PC(a,b) = \frac{\sum_{i \epsilon U_{ab}} (R_{ai} - \overline{R_a}) \times (R_{bi} - \overline{R_b})}{\sqrt{\sum_{i \epsilon U_{ab}} (R_{ai} - \overline{R_a})^2} \times \sqrt{\sum_{i \epsilon U_{ab}} (R_{bi} - \overline{R_b})^2}} \quad (4)$$

Pada persamaan (4), $U_{ab}$ merupakan sekelompok pengguna yang melakukan *rating* terhadap benda a dan b. $R_{ai}$ dan $R_{bi}$ merupakan nilai dari pengguna ke-i untuk benda a dan benda b. $\overline{R_a}$ dan $\overline{R_b}$ menentukan nilai rata-rata yang diterima oleh benda a dan benda b.

Perhitungan kesamaan dari setiap *item* dengan *item* lain yang berada dalam *cluster* yang sama dapat dilihat pada persamaan (5).

$$Similarity_i = \sum_{j=1, j \neq 1}^{n_s} PC(i,j) \quad (5)$$

Pada persamaan (5), $n_s$ adalah jumlah *item* dalam *cluster* yang diinginkan dan PC(i,j) adalah tingkat kesamaan antara antara dua *item* i dan j dalam *cluster*. Kepala *cluster* dipilih dengan menggunakan persamaan (6).

$$ClusterHead = \max_{1 \leq i \leq n_s} \{Similarity_i\} \quad (6)$$

Kesamaan anggota *cluster* dengan kepala *cluster* yang dipilih mempengaruhi kesesuaian *cluster* kromosom. Oleh karena itu, kesamaan setiap anggota *cluster* dengan kepala *cluster* dihitung denghan menggunakan persamaan (7).

$$Resemblance_i = PC(i, ClusterHead) \quad (7)$$

Pada setiap cluster, rata-rata kesamaan *item* dengan kepala *cluster* (μ atau dibaca mu) dihitung sesuai dengan persamaan (8).

$$\mu = \frac{1}{n_s - 1} \sum_{i=1}^{n_s - 1} Resemblance_i \quad (8)$$

Kesamaan setiap anggota dengan kepala *cluster* tidak boleh jauh dari rata-rata kesamaan, dan dispersi yang lebih kecil di sekitar titik rata-rata harus diperhatikan. Faktor ini dapat diperoleh dengan menghitung standar deviasi (σ atau dibaca sigma) dari kesamaan melalui persamaan (9).

$$\sigma = \sqrt{\frac{1}{n_s - 1} \sum_{i=1}^{n_s - 1} (Resemblance_i - \mu)^2} \quad (9)$$

Bobot dari parameter rata-rata dan standar deviasi tidak boleh sama dalam perhitungan *fitness function*. Secara umum, rata-rata memiliki bobot lebih besar dibandingkan dengan standar deviasi. Untuk mendapatkan rasio yang tepat antara keduanya dapat menguji jumlah yang berbeda (koefisien a dan b yang terdapat pada persamaan (10)). *Fitness function* dari setiap *cluster* ditentukan oleh persamaan (10).

$$ClusterFitness_i = a\mu_i + b\sigma_i \quad (10)$$

Rata-rata *fitness* dari *cluster* pada setiap kromosom dapat ditentukan dengan menggunakan persamaan (11).

$$ChromosomeFitness = \frac{1}{s} \sum_{i=1}^{s} ClusterFitness_i \quad (11)$$

Pada persamaan (11), s merupakan jumlah *cluster* yang terdapat pada kromosom yang diinginkan (Mohammadpour et al., 2019)

Sebuah *fitness function* harus mengembalikan nilai yang lebih tinggi untuk kondisi yang lebih baik(Russell & Norvig, 2010). GA sederhana yang menghasilkan hasil yang baik dalam banyak masalah praktis terdiri dari tiga operator:

***a. Selection***

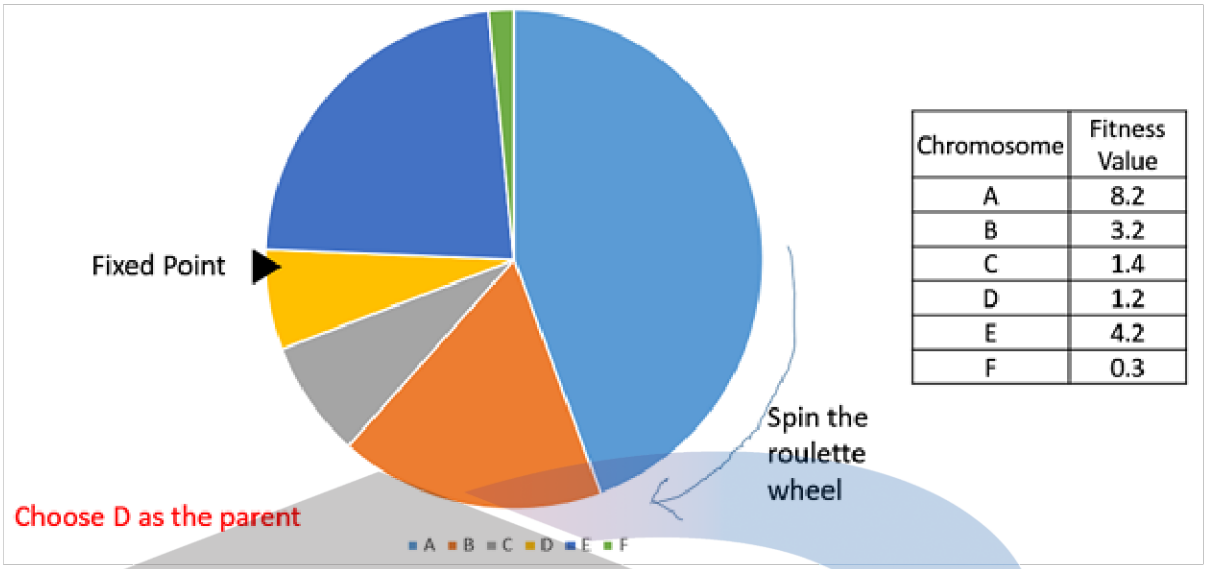
*Selection* adalah pemilihan individu dalam populasi yang akan menghasilkan keturunan untuk generasi berikutnya. Tujuan dari *selection* adalah menyisihkan individu dengan *fitness* yang lebih baik agar keturunan berikutnya memiliki kemungkinan mendapatkan *fitness* yang baik juga (sitasi disini). Terdapat beberapa metode seleksi yang dapat dilakukan seperti:

**1. *Roulette Wheel dan Stochastic Universal Sampling (SUS)***

Penerapan seleksi *roulette wheel* secara umum adalah dengan memberikan proporsi dari tiap kromosom yang didapatkan dengan membagi *fitness* kromosom dengan total *fitness* pada populasi. Tiap kromosom diberikan "potongan" dari *roulette wheel* yang proporsional terhadap *fitness* dari kromosom (Melanie, 1998).

*Roulette Wheel* kemudian akan dijalankan sebanyak *N* kali untuk memilih sebanyak *N* induk. Setiap kali *Roulette wheel* dijalankan, individu yang berada dibawah penanda (*Marker*) akan dipilih sebagai induk untuk generasi berikutnya (Melanie, 1998). Metode ini dapat diimplementasikan sebagai berikut:

1. Jumlahkan semua *fitness* pada populasi, kita akan sebut angka ini sebagai *Total.*
2. Pilih nilai acak yaitu *Rand* antara 0 sampai dengan *Total*.
3. Lakukan *looping* terhadap individu pada populasi, jumlahkan semua *fitness,* apabila jumlah *fitness* telah lebih besar atau sama dengan *Rand.* Maka individu tersebut akan dipilih sebagai induk dari hasil *roulette wheel*.




| Chromosome | Fitness Value |
|---|---|
| A | 8.2 |
| B | 3.2 |
| C | 1.4 |
| D | 1.2 |
| E | 4.2 |
| F | 0.3 |

Gambar 2.2 *Ilustrasi roulette wheel*

(Sumber: https://www.tutorialspoint.com/genetic_algorithms/images/roulette_wheel_selection.jpg)

*Stochastic Universal Sampling (SUS)* memiliki prinsip kerja yang hampir sama dengan *roulette wheel*. Ketika *roulette wheel* dijalankan sebanyak *N* kali untuk memilih *N* induk. SUS menjalankan roda sebanyak satu kali, namun dengan penanda (*Marker*) sebanyak *N* untuk memilih *N* induk (Melanie, 1998).

SUS dapat dilakukan sesuai potongan kode berikut (ditulis dalam bahasa C):

```
ptr = Rand(); /* Returns random number uniformly distributed in [0,1] */
for (sum = i = 0; i < N; i++)
      for (sum += ExpVal(i,t); sum > ptr; ptr++)
      Select(i);
```

Pada potongan kode diatas, *i* adalah indeks dari kromosom dan ExpVal(*i*,*t*) adalah *fitness* individu indeks *i* pada waktu *t*.

Walaupun demikian, SUS tidak menyelesaikan masalah yang timbul dari seleksi yang berdasar pada *fitness*. Walaupun variansi populasi masih tinggi di tahap awal, apabila sebagian kecil populasi memiliki *fitness* yang jauh lebih tinggi daripada yang lain. Maka pada seleksi yang berdasarkan *fitness*, kromosom dengan *fitness* tinggi dan keturunannya akan bertumbuh dengan sangat cepat sehingga mengurangi variansi pada populasi. Karena semua kromosom akan menjadi sangat mirip dan variansi *fitness* rendah, ruang pencarian GA akan berkurang drastis sebagai akibat variansi yang kecil tersebut. Fenomena ini dikenal sebagai "Konvergensi Prematur" (*Premature Convergence*) (Melanie, 1998).

**2. *Elitism***

*Elitism* memaksa GA untuk mempertahankan beberapa kromosom dengan *fitness* yang lebih baik di tiap generasi. Kromosom yang dipertahankan ini tidak boleh diubah sama sekali pada generasi yang baru. Kromosom ini dapat hilang apabila tidak terpilih sebagai induk atau dihancurkan oleh mutasi. Banyak peneliti yang mengemukakan bahwa *elitism* secara signifikan dapat meningkatkan performa GA (Melanie, 1998).

**3. *Boltzmann Selection***

*Boltzmann selection* adalah metode seleksi yang mirip dengan *simulated annealing*. Pada seleksi *boltzmann* temperatur yang berubah akan mengendalikan peluang seleksi. Pada temperatur yang tinggi, semua kromosom memiliki peluang yang sama untuk terpilih. Seiring menurunnya temperatur, kemungkinan terpilihnya kromosom dengan *fitness* yang lebih tinggi akan meningkat dan kemungkinan terpilihnya kromosom dengan *fitness* yang rendah akan menurun. Metode ini memungkinkan GA untuk memperkecil area pencarian terhadap kromosom yang lebih baik dan tetap mempertahankan tingkat variansi (Melanie, 1998).

**4. *Rank Selection***

*Rank selection* adalah metode alternatif yang bertujuan mencegah konvergensi prematur. Pada versi yang diajukan oleh Baker (1985), kromosom pada populasi diberikan peringkat berdasarkan *fitness*, dan nilai yang diharapkan pada tiap kromosom ditentukan oleh peringkat tersebut dan bukan berdasar secara absolut pada *fitness*. Pada metode ini tidak perlu dilakukan skala proporsi fitness. *Rank selection* menghindari sekelompok kecil kromosom dengan *fitness* yang tinggi mendominasi keturunan pada generasi berikutnya (Melanie, 1998).

*Rank selection* memiliki kekurangan yaitu memperlambat pencarian, karena hampir semua kromosom memiliki peluang terpilih yang sama. GA akan lebih lambat dalam menemukan kromosom dengan *fitness* yang paling baik.

**5. *Tournament Selection***

Pada *tournament selection*, dua kromosom akan dipilih secara acak dari populasi. Kemudian angka acak *rand* akan dihasilkan antara 0 sampai dengan 1. Jika *rand* < *k* (k adalah parameter, contohnya 0.75), maka kromosom dengan *fitness* yang lebih tinggi akan terpilih sebagai induk; sebaliknya, jika *rand* > *k* maka kromosom dengan *fitness* yang lebih rendah

yang akan terpilih. Kedua kromosom kemudian dikembalikan kedalam populasi dan dapat dipilih kembali (Melanie, 1998).

Dalam metode yang diusulkan, seleksi induk dalam populasi primer dilakukan menggunakan *roulette wheel*, yang berdasarkan pada probabilitas seleksi. Oleh karena itu, probabilitas pemilihan setiap kromosom diperoleh berdasarkan kesesuaiannya melalui persamaan (12).

$$Probability_k = \frac{ChromosomeFitness_k}{\sum_{i=1}^{n} ChromosomeFitness_i} \quad (12)$$

Jika *variance fitness* kromosom tinggi, kromosom dengan *fitness* tingkat tinggi akan memiliki peluang tinggi untuk menerapkan *crossover* dan metode yang diusulkan akan memiliki tingkat konvergensi yang tinggi dan akan terjebak dalam *local optimum*. Ini berarti bahwa metode *roulette wheel* tidak dapat membangun keseimbangan yang tepat antara tingkat konvergensi dan keragaman populasi dalam hal *fitness*. Untuk mengatasi masalah ini, kita dapat menggunakan metode Boltzmann yang dihasilkan dari pemanasan logam, atau metode *simulated annealing*. Pada awalnya, pada suhu tinggi, semua kromosom memiliki probabilitas yang sama untuk dipilih. Secara bertahap, dengan penurunan suhu, peluang untuk memilih kromosom dengan *fitness* tinggi meningkat sementara peluang memilih kromosom dengan *fitness* rendah menurun. Probabilitas memilih setiap kromosom sesuai dengan metode *simulated annealing* adalah seperti yang ditunjukkan pada persamaan (13).

$$SAP_k = Probability_k - \left(T * \left(Probability_k - \frac{1}{n}\right)\right) \quad (13)$$

Pada persamaan (13), T adalah nilai suhu dan $n$ adalah jumlah kromosom dalam populasi. Nilai-nilai T berada dalam interval 0 dan 1, sehingga nilai awal sama dengan 1, dan setiap kali setelah sejumlah iterasi tertentu, ia dikurangi pada laju 0,1. Untuk menerapkan metode *roulette wheel*, probabilitas kumulatif dari masing-masing kromosom diperoleh berdasarkan metode *simulated annealing* sesuai dengan persamaan (14) (Mohammadpour et al., 2019)

$$CumulativeProbability_k = \sum_{i=1}^{k} SAP_i \quad (14)$$

***b. Crossover***

*Crossover* atau rekombinasi adalah operator genetik yang digunakan untuk menggabungkan informasi genetik induk untuk menghasilkan keturunan baru. Ini adalah salah satu cara secara stokastik untuk menghasilkan solusi baru dari populasi yang ada dan analog dengan *crossover* yang terjadi selama reproduksi seksual dalam biologi. Solusi juga dapat dihasilkan dengan mengkloning solusi yang ada dan yang analog dengan reproduksi aseksual. Solusi yang baru dihasilkan biasanya bermutasi sebelum ditambahkan ke populasi. *Crossover*

dibuat dengan harapan bahwa kromosom baru akan memiliki bagian yang baik dari kromosom lama dan mungkin kromosom baru akan lebih baik (Goldberg, 1989).

*Crossover* yang paling sederhana adalah *crossover* satu titik (*Single point crossover*). Pada penerapannya, *crossover* satu titik akan mencari satu titik *crossover* secara acak, kemudian bagian dari kedua induk setelah titik *crossover* akan ditukar untuk menghasilkan dua keturunan. *Crossover* satu titik memiliki kelemahan yaitu tidak bisa menghasilkan semua skema yang dimungkinkan. Sebagai contoh, tidak mungkin untuk menggabungkan 11*****1 dengan ****11** dan menghasilkan keturunan 11**11*1 (Melanie, 1998). Untuk melakukan kombinasi terhadap skema yang lebih luas, dapat diterapkan *crossover* dua titik.

*Crossover* dua titik digunakan untuk *crossover* dalam metode yang diusulkan. Dalam hal ini, panjang pasangan kromosom yang dipilih untuk operasi *crossover* dapat bervariasi, karena jumlah *cluster* yang berbeda. Setelah operasi *crossover*, jika di bagian item dari kromosom anak, item dibuat dengan nomor *cluster* yang tidak ada di kromosom, sebuah *cluster* dengan angka s + 1 (s adalah jumlah *cluster* dari kromosom yang sesuai) akan ditambahkan dan nomor *cluster item* dan semua *item* serupa akan sama dengan s +1. Proses ini dapat diikuti untuk *item* lain dengan nomor *cluster* lain yang tidak ada di bagian *cluster* kromosom (Mohammadpour et al., 2019)

***c. Mutation***

*Mutation* atau mutasi adalah perubahan acak sesekali dari nilai posisi *string*. Ketika digunakan bersamaan dengan *selection* dan *crossover*, *mutation* merupakan jaminan terhadap konvergensi prematur atas gagasan-gagasan penting. *Mutation* dibuat untuk mencegah GA jatuh ke lokal ekstrim. *Selection*, *crossover*, dan *mutation* telah terbukti sangat sederhana dan efektif dalam menyelesaikan sejumlah masalah optimisasi penting (Goldberg, 1989).

Dalam metode yang diusulkan, mutasi dilakukan dengan penggabungan *cluster*. Pada metode ini, dua *cluster* dipilih secara acak dan *cluster* dengan angka yang lebih besar (*cluster* awal) akan ditransfer dan digabungkan dengan *cluster* dengan angka yang lebih kecil (*cluster* tujuan) dan *cluster* awal akan dihapus. Setelah penggabungan, semua *cluster* dengan angka yang lebih kecil dari *cluster* awal akan dikurangi dengan angka satu.

### 2.3.2 Simulated Annealing

*Simulated Annealing* (SA) adalah teknik probabilistik untuk mendekati optimum global dari fungsi yang diberikan. Khususnya, SA adalah *metaheuristik* untuk memperkirakan

optimasi global dalam ruang pencarian besar untuk masalah optimasi. SA dapat mendekati global maksimum, dengan biaya perhitungan yang lebih besar (Russell & Norvig, 2010).

SA juga adalah salah satu *metaheuristik* paling sederhana dan metode paling terkenal untuk mengatasi masalah optimisasi global *black box* yang sulit, yang tujuannya fungsi tidak diberikan secara eksplisit dan hanya dapat dievaluasi melalui beberapa simulasi komputer yang dikarenakan biaya yang sangat mahal. SA banyak diimplementasikan dalam aplikasi dalam kehidupan nyata. Salah satu fitur utama dari SA adalah kemampuannya untuk menerima transisi yang menurunkan fungsi tujuan (Nikolaev & Jacobson, 2010).

Diberikan struktur lingkungan, SA dapat dilihat sebagai algoritma yang terus-menerus berupaya mengubah konfigurasi saat ini menjadi salah satu tetangganya. Mekanisme ini secara matematis paling baik dijelaskan dengan menggunakan rantai Markov yaitu urutan uji coba, di mana hasil dari setiap uji coba hanya bergantung pada hasil yang sebelumnya. Rantai Markov dijelaskan melalui serangkaian probabilitas kondisional $P_{ij}$(k-1,k) untuk setiap pasangan hasil (i,j). Dengan $P_{ij}$(k-1,k) adalah probabilitas bahwa hasil dari uji coba ke-k adalah i, mengingat bahwa hasil dari uji coba ke k-1 adalah i. Biarkan $a_i$(k) menunjukkan probabilitas hasil i pada percobaan-k, maka $a_i$(k) diperoleh dengan menyelesaikan rekursi pada persamaan (15).

$$a_i(k) = \sum_l a_i(k-1)\,.\,P_{li}(k-1,k)\,,\quad k = 1,2,\ldots, \tag{15}$$

Dengan jumlahnya diambil atas semua hasil yang mungkin. Selanjutnya, X(k} menunjukkan hasil dari uji coba ke-k yang dapat dilihat pada persamaan (16) dan (17).

$$P_{ij}(k-1,k) = \Pr\{X(k) = j \mid X(k-1) = i\} \tag{16}$$

dan

$$a_i(k) = \Pr\{X(k) = i\} \tag{17}$$

Jika probabilitas bersyarat tidak bergantung pada k, rantai Markov yang sesuai disebut *homogenous*, jika tidak disebut *inhomogenous*.

Dalam kasus *Simulated Annealing*, probabilitas bersyarat $P_{ij}$(k -1, k) menunjukkan probabilitas bahwa transisi ke-k adalah transisi dari konfigurasi i ke konfigurasi i. Jadi, X (k} adalah konfigurasi yang diperoleh setelah transisi k. Dalam pandangan ini, $P_{ij}$(k -1, k) disebut probabilitas transisi dan |R| x |R|-matriks P(k - 1, k) matriks transisi.

Probabilitas transisi tergantung pada nilai parameter kontrol c, analog dari suhu dalam proses anil fisik. Jadi, jika c dijaga konstan, rantai Markov yang sesuai adalah homogen dan matriks transisinya P = P(c) dapat didefinisikan melalui persamaan (18).

$$P_{ij}(c) = \begin{cases} G_{ij}(c)A_{ij}(c) & \forall j \neq i \\ 1 - \sum_{l=1,l\neq i}^{|R|} G_{il}(c)A_{il}(c) & j=i \end{cases} \tag{18}$$

Setiap probabilitas transisi didefinisikan sebagai produk dari dua probabilitas kondisional berikut yaitu probabilitas generasi $G_{ij}(c)$ menghasilkan konfigurasi i dari konfigurasi i, dan probabilitas penerimaan $A_{ij}(c)$ untuk menerima konfigurasi i, setelah itu telah dihasilkan dari i. Matriks yang sesuai G(c) dan A(c) masing-masing disebut matriks generasi dan penerimaan. P(c) adalah matriks stokastik yaitu Vi: Ej Pij (C} = 1.

Seperti yang ditunjukkan sebelumnya, parameter kontrol c menurun selama algoritma. Sehubungan dengan penurunan ini, dua formulasi algoritma dapat dibedakan sebagai berikut:

a. Algoritma *homogeneous*: algoritma ini dijelaskan oleh urutan rantai Markov yang homogen. Setiap rantai Markov dihasilkan pada nilai c dan c yang menurun di antara rantai Markov berikutnya.
b. Algoritma *inhomogeneous*: algoritma ini dijelaskan oleh rantai tunggal Markov yang tidak homogen. Berkurangnya nilai dari c karena berada di antara transisi selanjutnya.

Algoritma *simulated annealing* memperoleh global minimum jika telah mencapai sejumlah transisi, katakanlah K, hubungan yang berlaku dapat dilihat pada persamaan (19).

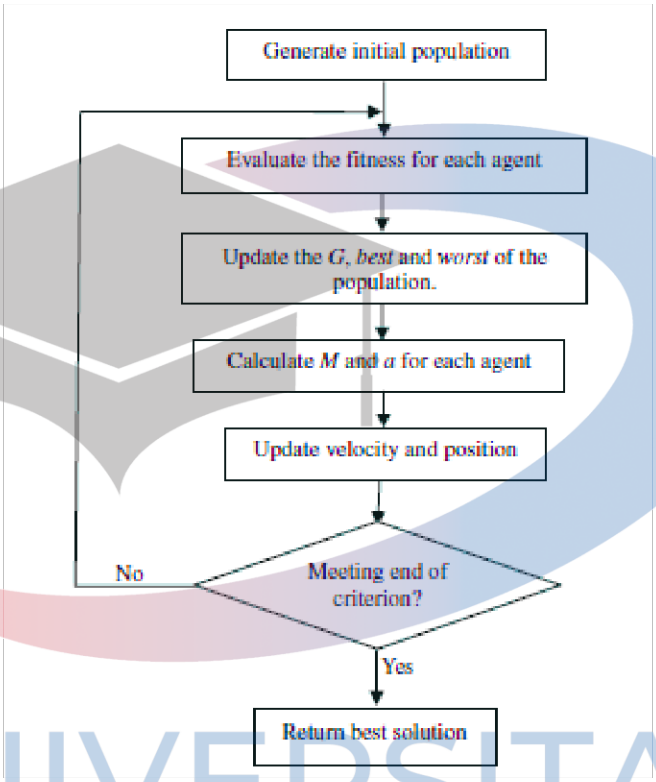
$$P_r\{X(K) \in R_{opt}\} = 1 \quad (19)$$

Dengan $R_{opt}$ adalah set konfigurasi minimal global (van Laarhoven & Aarts, 1987).

### 2.3.3 Gravitational Search Algorithm

*Gravitional Search Algorithm* (GSA) adalah teknik heuristik di bidang numerik optimasi. Ini adalah pencarian berbasis *scholastic* dan pencarian *swarm-based* untuk masalah-masalah kombinasi yang rumit. GSA adalah algoritma optimasi heuristik berdasarkan hukum gerak *Newton* dan hukum gravitasi. Itu terdiri massa (agen) di mana massa yang lebih berat dianggap sebagai solusi yang menonjol dari masalah tersebut. Setiap algoritma heuristik mengikuti ekspor dan kriteria eksploitasi. Serupa dalam GSA, algoritma pertama-tama menjelajahi wilayah pencarian kemudian putaran iterasi. Ini disebut konvergen ke solusi yang disebut langkah eksploitasi. GSA adalah algoritma yang berkembang sangat cepat yang membantu menemukan solusi optimal atau mendekati optimal (Bala & Yadav, 2019).

Dalam GSA, setiap massa (agen) memiliki empat spesifikasi, yaitu posisi, massa inersia, massa gravitasi aktif, dan massa gravitasi pasif. Posisi massa sesuai dengan solusi masalah, massa gravitasi dan inersia ditentukan menggunakan *fitness function*. Dengan kata lain, setiap massa menyajikan solusi dan algoritma dinavigasi dengan menyesuaikan massa gravitasi dan inersia dengan tepat. Pada selang waktu, diharapkan massa akan tertarik oleh massa terberat. Massa ini akan menghadirkan solusi optimal di ruang pencarian (Rashedi et al., 2009).




Gambar 2.3 *General Principle of GSA*

Sumber: (Rashedi et al., 2009)

Sekarang, pertimbangkan sebuah sistem dengan N agen (massa). Posisi agen ke-i dapat ditentukan dengan menggunakan persamaan (20).

$$X_i = \left(x_i^1, \dots, x_i^d, \dots, x_i^n\right) \quad for\ i = 1,2, \dots, N \tag{20}$$

Dengan $x_i^d$ menyajikan posisi dari agen ke-i dalam dimensi ke-d. Karena gravitasi hukum, gaya tarik-menarik antara massa i dan j pada waktu t dan dimensi d dapat dihitung dengan rumus yang terdapat pada persamaan (21).

$$F_{ij}^d(t) = G(t) \cdot \frac{M_i(t) \times M_j(t)}{R_{ij}(t) + \varepsilon} \left(x_j^d(t) - x_i^d(t)\right) \tag{21}$$

Dengan $M_j$ adalah massa gravitasi aktif yang terkait dengan agen j, $M_i$ adalah massa gravitasi pasif yang terkait dengan agen i, G(t) adalah konstanta gravitasi pada waktu t, e adalah

konstanta kecil, dan $R_{ij}$ (t) adalah jarak Euclidian antara dua agen i dan j yang dapat dihitung dengan menggunakan persamaaan (22).

$$R_{ij}(t) = \left\| X_i(t), X_j(t) \right\|_2 \quad (22)$$

G(t) adalah konstanta gravitasi yang mengontrol proses menggunakan variabel α (dibaca *alpha*) dan berkurang seiring waktu yang dapat dihitung dengan menggunakan persamaan (23).

$$G(t) = G_0 \times \exp(-\alpha \times iter/\max iter) \quad (23)$$

Karenanya, gaya total massa i pada waktu t dapat dihitung dengan menggunakan persamaan (24).

$$F_i^d(t) = \sum_{j=1, j\neq i}^{N} rand_j F_{ij}^d(t) \quad (24)$$

$rand_j$ mewakili angka acak dalam interval [0, 1]. Dengan Hukum gerak Newton, akselerasi objek i dalam dimensi d dapat dihitung dengan menggunakan persamaan (25).

$$ac_i^d(t) = \frac{F_i^d(t)}{M_i(t)} \quad (25)$$

Jika semua agen menarik satu sama lain yang menyebabkan gerakan global ke arah objek dengan massa yang berat, maka posisi partikel dipengaruhi oleh kecepatan ($vel_i$) dapat dilihat pada persamaan (26) dan akselerasi $ac_i$ pada persamaan (27).

$$vel_i^d\,(t+1) = rand_i \times vel_i^d\,(t) + ac_i^d\,(t) \quad (26)$$

dan

$$x_i^d\,(t+1) = x_i^d\,(t) + vel_i^d\,(t+1) \quad (27)$$

Dengan bantuan persamaan kecepatan dan posisi, kita dapat memperbarui posisi agen. Ini membantu untuk memindahkan massa ke arah massa yang lebih berat yang dianggap sebagai solusi yang menonjol. Setelah berjalan akan ditentukan iterasi atau selang waktu, semua massa berkumpul menjadi massa yang lebih berat yang mengikuti solusi optimal. Untuk memperbarui agen dapat menggunakan persamaan (28).

$$m_i(t) = \frac{fit_i(t) - worst(t)}{best(t) - worst(t)}, M_i(t) = \frac{m_i(t)}{\sum_{j=1}^{N} m_j(t)} \quad (28)$$

Dengan $fit_i(t)$ mewakili nilai kesesuaian objek i dan terbaik (t) yang dapat dilihat pada persamaan (29) dan terburuk (t) diberikan untuk kasus maksimalisasi yang dapat dilihat pada persamaan (30) (Rashedi et al., 2009)

$$best(t) = \max_{i \in (1,2,\ldots N)} fit_i(t) \quad (29)$$

dan

$$worst(t) = \min_{i \in (1,2,\ldots N)} fit_i(t) \quad (30)$$

### 2.3.4 Gravitational Emulation Local Search (GELS) Algorithm

Algoritma ini memperkenalkan konsep pengacakan dengan dua dari empat parameter utama yaitu kecepatan dan gravitasi dalam fisika melalui *swapping* dalam hal kelompok dengan menggunakan angka acak dalam pencarian lokal yang ada untuk menghindari lokal minimum. GELS mengambil prinsip-prinsip alami daya tarik gravitasi sebagai dasarnya. GELS menghitung gaya gravitasi antara solusi atau objek di ruang pencarian dengan dua metode. Metode pertama, solusi dari ruang lingkungan lokal dipilih sebagai solusi saat ini dan gaya gravitasi antara dua solusi ini dapat dihitung. Metode kedua menerapkan rumus untuk semua solusi dalam lingkungan dan melacak gaya gravitasi antara masing-masing dan solusi saat ini secara individual semua solusi (Barzegar et al., 2009)

Algoritma GELS termasuk objek *pointer* yang dapat bergerak melalui ruang pencarian dan massa yang dimaksudkan untuk objek *pointer* stabil di seluruh perhitungan dan objek ini mengacu pada solusi dengan massa terbesar. Algoritma akan berakhir ketika salah satu dari dua kondisi berikut terjadi yaitu, semua elemen dalam vektor kecepatan awal telah nol atau jumlah iterasi maksimum yang diijinkan telah selesai (Barzegar et al., 2009).

Menurut penelitian terdahulu, algoritma hibrida GA dan GELS menghasilkan hasil yang lebih baik walaupun dengan menghabiskan waktu yang lebih lama dibandingkan metode clustering lainnya (Mohammadpour et al., 2019). Hasil penelitian terdahulu dapat dilihat pada tabel 2.1

Tabel 2.1 Hasil penelitian terdahulu

| | PCA-GAKM | PCA-SOM | UPCC | GAKM-cluster | PCA-K-Means | K-Means Cuckoo | GA-GELS |
|---|---|---|---|---|---|---|---|
| **MAE** | 0.94 | 0.98 | 0.80 | 0.76 | 0.92 | 0.68 | 0.66 |
| **RMSE** | 1.20 | 1.25 | 1.02 | 0.97 | 1.17 | 0.87 | 0.84 |
| **Waktu (detik)** | 29.32 | 147.73 | 179.34 | 325.71 | 65.56 | 63.22 | 419.81 |

Algoritma GELS menganggap kromosom yang dikirim sebagai objek yang memiliki massa. Tiap kromosom memiliki massa aktif, pasif dan intersial, ketiga massa tersebut bernilai sama dengan massa kromosom (*chromosome mass*). Massa kromosom disesuaikan dengan nilai *fitness* kromosom. Massa awal kromosom yang dikirimkan pada waktu *t* didapatkan melalui persamaan (31).

$$PrimaryMass_i(t) = \frac{ChromosomeFitness_i(t) - Worst(t)}{Best(t) - Worst(t)} \quad (31)$$

Pada persamaan (30), *Best(t)* dan *Worst(t)* masing-masing adalah nilai *fitness* tertinggi dan terendah pada populasi yang dikirimkan ke GELS pada waktu *t*.

Sekarang, massa awal kromosom akan dinormalkan dengan persamaan (32) untuk mendapatkan massa kromosom pada waktu *t*.

$$ChromosomeMass_i(t) = \frac{PrimaryMass_i(t)}{\sum_{j=1}^{N} PrimaryMass_j(t)} \quad (32)$$

Jarak Euclidean digunakan untuk menentukan jarak antara massa dalam algoritma GELS. Dengan demikian, jarak antara dua massa i dan j pada waktu t didapatkan dengan menggunakan persamaan (33).

$$Distance_{ij}(t) = \left\| x_i(t).x_j(t) \right\|_2 \quad (33)$$

Perbedaan antara dua massa i dan j dalam dimensi d pada waktu t adalah jumlah item yang berbeda antara cluster ke-d mereka, sehingga untuk menghitung jarak antara massa i dan j dalam dimensi d pada waktu t dapat digunakan jarak *Jaccard* sesuai dengan persamaan (34).

$$Distance_{ij}^{d}(t) = 1 - \frac{\left| x_i^d \cap x_j^d \right|}{\left| x_i^d \cup x_j^d \right|} \quad (34)$$

Oleh karena itu, menurut persamaan (33) dan persamaan (34), jarak Euclidean antara dua massa i dan j pada waktu t dapat dihitung dengan menggunakan persamaan (35).

$$Distance_{ij}(t) = \sqrt{\sum_{d=1}^{s} \left( Distance_{ij}^{d}(t) \right)^2} \quad (35)$$

Hasil dari gaya gravitasi yang diterapkan oleh tetangga untuk solusi saat ini sekarang dihitung. Gaya gravitasi diterapkan pada solusi saat ini oleh massa ke-j dalam dimensi d pada waktu t dihitung melalui persamaan (36).

$$F_{CS\,j}^{d}(t) = G(t) \frac{M_{Passive\,CS}(t) \times M_{Active\,j}(t)}{Distance_{CS\,j}(t) + \varepsilon} \left( x_j^d(t) - x_{CS}^d(t) \right) \quad (36)$$

$G(t)$, $M_{Passive\ CS}(t)$, $M_{Active\ j}(t)$ dan $Distance_{CSj}(t)$ masing-masing adalah konstanta gravitasi, massa gravitasi pasif dari solusi saat ini, massa gravitasi aktif dari massa j dan jarak Euclidean antara solusi saat ini dan massa ke-j pada waktu t. ε juga merupakan konstanta kecil yang digunakan untuk mencegah pembagian dengan kesalahan nol. Persamaan eksponensial (37) digunakan untuk secara bertahap mengurangi konstanta gravitasi dari waktu ke waktu.

$$G(t) = G_o e^{-\alpha \frac{t}{T}} \quad (37)$$

$G_o$ dan α adalah nilai utama dari konstanta gravitasi dan masing-masing merupakan nilai konstan yang positif. t merupakan jumlah iterasi sekarang dan T adalah jumlah dari seluruh iterasi yang ada. Gaya yang diterapkan oleh semua massa tetangga ke solusi saat ini dalam dimensi d pada waktu t dihitung dengan menggunakan persamaan (38).

$$F_{CS}^{d}(\mathrm{t}) = \sum_{j=1}^{K} rand_j \, F_{CS\,j}^{d}(\mathrm{t}) \quad (38)$$

Pada persamaan (38), *rand*$_j$ merupakan angka acak dengan distribusi yang sama dalam interval 0 dan 1 yang digunakan untuk menjaga karakteristik keacakan dari pencarian. Percepatan dari solusi sekarang dalam dimensi d pada waktu t dapat dihitung dengan menggunakan persamaan (39).

$$a_{CS}^{d}(t) = \frac{F_{CS}^{d}(t)}{M_{inertia\ CS}(t)} \quad (39)$$

$M_{inertia\ CS}(t)$ adalah massa inersia dari solusi sekarang pada waktu t. persamaan (39) juga dapat ditulis ulang menjadi persamaan (40).

$$a_{CS}^{d}(t) = G(t) \sum_{j=1, j \neq CS}^{K} \left[ rand_j \frac{ChromosomeMass_j(t)}{Distance_{CS\,j}(t) + \varepsilon} \left( x_j^d(t) - x_{CS}^d(t) \right) \right] \quad (40)$$

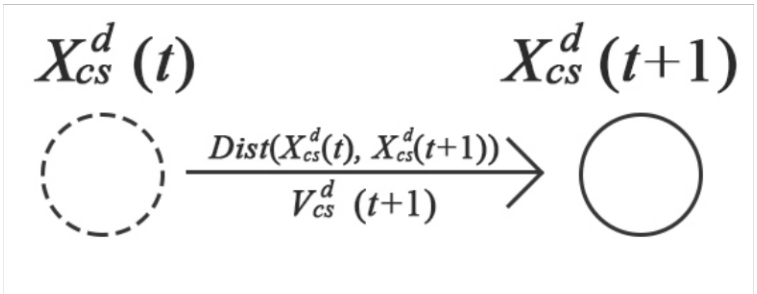
Kecepatan baru dari solusi saat ini sama dengan jumlah koefisien dari kecepatan saat ini dan percepatan arus, yang dalam dimensi d, sesuai dengan persamaan (41) (Mohammadpour et al., 2019) .

$$v_{CS}^{d}(t+1) = rand_{CS} \, . \, v_{CS}^{d}(t) + a_{CS}^{d}(t) \quad (41)$$

Solusi saat ini bergerak sesuai dengan lokasi saat ini dan kecepatan barunya pada dimensi d dan diletakkan pada lokasi baru pada dimensi d. Pendekatan yang dilakukan biasanya adalah dengan menjumlahkan posisi sekarang dengan kecepatan baru sesuai persamaan (26) yaitu $x_i^d\ (t+1) = x_i^d\ (t) + \ vel_i^d\ (t+1)$. Namun karena posisi $x_i^d$ ditentukan oleh jumlah *item* pada *cluster*, maka operator penjumlahan harus disubstitusi (Dowlatshahi & Nezamabadi-Pour, 2014). Posisi baru cluster haruslah hampir sesuai dengan persamaan (42)

$$Dist_j\left(x_{cs}^d\ (t+1),\ x_{cs}^d\ (t)\right) \approx v_{CS}^d(t+1) \quad (42)$$

Pada persamaan (41) "≈" menandakan "hampir sama dengan" dan digunakan untuk menggantikan "=" karena adalah tidak mungkin untuk menghasilkan cluster baru pada posisi $x_{cs}^d\ (t+1)$ sehingga nilai $Dist_j\left(x_{cs}^d\ (t+1),\ x_{cs}^d\ (t)\right)$ sama dengan $v_{CS}^d(t+1)$. Ketika nilai $v_{CS}^d(t+1)$ lebih besar dari 1 maka nilai harus disesuaikan menjadi 1 karena $Dist_j\left(x_{cs}^d\ (t+1),\ x_{cs}^d\ (t)\right) \leq 1$ (Dowlatshahi & Nezamabadi-Pour, 2014).




Gambar 2.4 Perpindahan lokasi $x_{cs}^{d}(t)$ menuju ke $x_{cs}^{d}(t+1)$

Dalam menghasilkan posisi baru pada $x_{cs}^{d}(t+1)$, jumlah *item* yang sama pada posisi $x_{cs}^{d}(t+1)$ dan $x_{cs}^{d}(t)$ harus mendekati nilai $v_{CS}^{d}(t+1)$. Faktanya, *item* yang sama antara posisi $x_{cs}^{d}(t+1)$ dan $x_{cs}^{d}(t)$ adalah bagian yang diwariskan dari solusi $x_{cs}(t)$ ke solusi $x_{cs}(t+1)$. Jumlah item yang diwariskan pada dimensi *d* dinotasikan dengan $n_{cs}^{d}(t+1)$ dan didapatkan dengan persamaan (43). Dengan $x_{cs}^{d}(t)$ menandakan jumlah *item* pada posisi $x_{cs}^{d}(t)$.

$$n_{cs}^{d}(t+1) = \left\lfloor \left(1 - v_{CS}^{d}(t+1)\right) \mid x_{cs}^{d}(t) \mid \right\rfloor \quad (43)$$

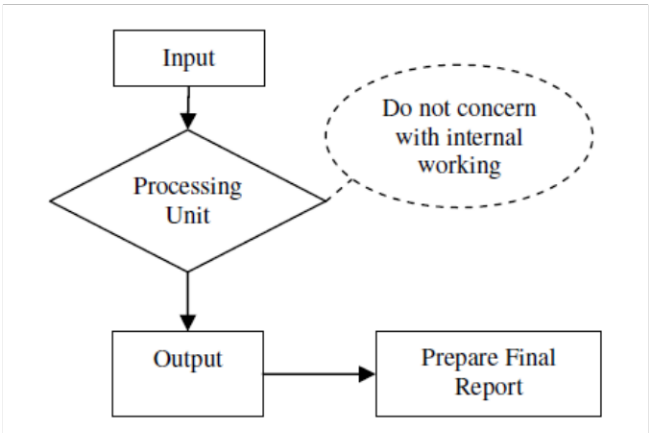
Setelah $n_{cs}^{d}(t+1)$ *item* acak diwariskan dari $x_{cs}^{d}(t)$ ke $x_{cs}^{d}(t+1)$. Maka sisa item yang belum dialokasikan ke $x_{cs}^{d}(t+1)$ akan dialokasikan berdasarkan *head* yang paling dekat.

## 2.4 Black Box Testing

Pengujian dimaksudkan untuk menunjukkan bahwa suatu program melakukan apa yang dimaksudkan untuk dilakukan dan untuk menemukan cacat program sebelum digunakan (Lewis, 2000).

Dua pendekatan dasar untuk *software testing* adalah *black box testing* dan *white box testing*. *White box testing* didasarkan pada analisis kerja internal dan struktur perangkat lunak. Ini hanya memeriksa bagaimana sistem memproses input untuk menghasilkan output yang diperlukan. Di sisi lain *black box testing* berfokus pada persyaratan fungsional perangkat lunak. *Black box testing* adalah bagian integral dari pengujian kebenaran tetapi idenya tidak terbatas pada pengujian kebenaran saja(Khan, 2011).

Pengujian *black-box* terjadi sepanjang siklus hidup pengembangan perangkat lunak dan siklus hidup pengujian perangkat lunak yaitu dalam *regression testing*, *acceptance testing*, *unit testing*, *integration testing* dan *system testing*. Jenis-jenis pengujian dalam teknik ini benar-benar fokus pada pengujian fungsionalitas aplikasi perangkat lunak (Khan, 2011).




Gambar 2.5 *Working process of black box testing technique*

Sumber: (Khan, 2011)

Gambar 2.7 menunjukkan langkah-langkah yang menjelaskan proses kerja pengujian *black-box*, dengan penjelasan sebagai berikut (Khan, 2011):

1. *Input*: Persyaratan dan spesifikasi fungsional sistem diperiksa. Dokumen desain tingkat tinggi dan kode sumber blok aplikasi juga diperiksa. Penguji memilih *input* yang valid dan menolak *input* yang tidak valid.
2. *Processing unit*: Dalam unit pemrosesan, penguji membuat kasus uji dengan *input* yang dipilih dan menjalankannya. Penguji juga melakukan *load testing*, *stress testing*, *security review* dan *globalization testing*. Jika ada kecacatan yang terdeteksi, maka harus segera diperbaiki dan diujikan kembali.
3. *Output*: Setelah pengujian-pengujian di atas sudah selesai, penguji mendapatkan keluaran yang diharapkan dan mempersiapkan laporan terakhir.

Dalam pengujian *black-box* atau pengujian fungsional, kondisi pengujian dikembangkan berdasarkan fungsionalitas program atau sistem. Artinya penguji memerlukan informasi tentang data input dan *output* yang diamati, tetapi tidak harus mengetahui cara kerja dari program atau sistem tersebut (Lewis, 2000)

Berikut adalah beberapa jenis model pengujian *black-box*:

*a. Boundary value testing*

Teknik *boundary value testing* merupakan pengujian *black-box* yang berfokus pada batas-batas kesetaraan kelas *input* dan *output*. Kesalahan-kesalahan cenderung berada pada batas-batas tersebut, sehingga jika pengujian berfokus pada area ini akan meningkatkan peluang untuk mendeteksi kesalahan yang ada (Lewis, 2000). Terdapat empat jenis *boundary value*

*testing* yaitu *normal boundary value testing*, *robust boundary value testing*, *worst-case boundary value testing*, dan *robust worst-case boundary value testing* (Jorgensen, 2013).

*b. Use-case testing*

Pengujian *use-case* digunakan untuk mengetahui persayaratan dari perangkat lunak. Spesifikasi *use-case* merupakan dokumen-dokumen yang dipersiapkan untuk mengetahui persyaratan yang lebih detail. Pengujian *use-case* biasanya digunakan untuk mengetahui persyaratan-persyaratan dalam metodologi desain dan analisis *object-oriented.* Spesifikasi *use-case* berisi informasi-informasi yang ditunjukkan pada Tabel 2.1 (Desai & Srivastava, 2016).

Tabel 2.2 *Test Case Table*

| | |
|---|---|
| **Nama** | |
| **Deskripsi** | |
| **Aktor** | |
| **Skenario Utama** | |
| **Kondisi Awal** | |
| **Aksi Aktor** | **Reaksi Sistem** |
| 1. | 2. |
| | 3. |
| 4. | 5. |
| **Kondisi Akhir** | 6. |
| **Kondisi Alternatif** | 1.1 |

Untuk setiap fitur yang akan diuji, masing-masing fitur membutuhkan satu tabel *test case*.

*c. Comparison testing*

*Comparison testing* digunakan untuk memverifikasi kinerja perangkat lunak dan variannya yang berbeda di bawah konfigurasi perangkat keras dan perangkat lunak yang berbeda. Dalam beberapa aplikasi, keandalan dari aplikasi itu sangat penting dan membuat kebutuhan aplikasi tersebut semakin meningkat sehingga perlu dilakukan *comparison testing* (Khannur, 2014).

*d. Performance testing*

Untuk sistem apapun, *performance testing* sangatlah penting. Tujuan dari *performance testing* yaitu untuk memeriksa apakah sistem mampu untuk mencapai standar dari spesifikasi kebutuhan dari segi performa (Khannur, 2014).

Keuntungan menggunakan *black box testing* yaitu penguji dapat berasal dari kalangan non-teknis, karena bagian dalam sistem tidak diuji. Penguji hanya perlu memahami fungsionalitas sistem dari dokumen spesifikasi persyaratan (Desai & Srivastava, 2016).

## 2.5 Metode Haversine

Metode Haversine digunakan untuk menemukan titik-titik terdekat dalam dalam radius tertentu. Untuk menghitung jarak lingkaran antar titik, diperlukan beberapa parameter yaitu *latitude*, *longitude*, *distance equatorial radius*, dan terakhir menentukan jarak maksimum untuk radius pencarian (Ochoa et al., 2012). Untuk menentukan jarak dengan menggunakan formula Haversine dapat dilihat pada persamaan (44).

$$d = 2R \arcsin(\sqrt{sin^2\left(\frac{\varphi_2 - \varphi_1}{2}\right) + \cos(\varphi_1)\, cos\,(\varphi_2)\, sin^2(\frac{\lambda_2 - \lambda_1}{2})}\,) \quad (44)$$

dengan d adalah adalah jarak antar titik, $\varphi_1$ adalah *latitude* dari titik 1, $\varphi_2$ adalah *latitude* dari titik 2, $\lambda_1$ adalah *longitude* dari titik 1, $\lambda_2$ adalah *longitude* dari titik 2, R adalah radius bumi (6,371 km), dan arcsin adalah *inverse* sin.

Untuk mengatasi masalah kinerja, sehingga persamaan (44) disederhanakan menjadi persamaan (45).

$$d = \text{acos}(\sin\varphi_1 \sin\varphi_2 + \cos\varphi_1 \cos\varphi_2 \cos\Delta\lambda)\, R \quad (45)$$

## 2.6 Teknologi yang Digunakan

### 2.6.1 Dagger 2

Dagger 2 merupakan salah satu *dependency injection framework* terbaik dalam komunitas android. Kelebihan lain dari Dagger 2 adalah bersifat *open source*. *Dependency injection* dianggap *best practice* dan membuat baris-baris kode program menjadi lebih terstruktur. Cara kerja dari Dagger 2 adalah menggunakan *annotation* untuk menghasilkan kode dan juga untuk mengakses *field.* Annotation yang digunakan di dalam Dagger 2 ada 3 yaitu (Karanpuria & Roy, 2018):

a. @Module dan @Provides: untuk menetapkan *class* dan *method* yang menyediakan *dependency*.
b. @Inject: untuk meminta *dependency* dan bisa digunakan dalam sebuah *constructor*, *field* ataupun dalam sebuah *method*.
c. @Component: mengaktifkan modul yang telah dipilih dan juga digunakan untuk menjalankan *dependecy injection*.

### 2.6.2 Retrofit 2

Retrofit[1] adalah REST *client* yang aman untuk Android dan Java yang bertujuan untuk mempermudah penggunaan layanan web REST. Retrofit 2 secara *default* memanfaatkan OkHttp sebagai lapisan jaringan dan dibangun di atasnya. Retrofit secara otomatis membuat serial respons JSON menggunakan POJO (*Plain Old Java Object*) yang harus didefinisikan sebelumnya untuk Struktur JSON.

### 2.6.3 Surprise

*Surprise*[2] adalah library untuk membangun dan menganalisa sistem rekomendasi berdasarkan data umpan balik eksternal yang berbasis pada *library scikit*. *Surprise* didesain untuk memberikan pengguna kendali penuh atas eksperimen terhadap sistem rekomendasi.

*Surprise* menyediakan algoritma prediksi seperti *baseline algorithm, neighborhood method,* prediksi berbasis *matrix factorization* (SVD, PMG, SVD++, NMF). *Surprise* juga menyediakan algoritma perhitungan kemiripan seperti *cosine,* MSD, dan korelasi pearson. Semua algoritma tersebut diintegrasikan kedalam paket *surprise* dan siap untuk digunakan, didukung dengan *tool built-in* untuk mengevaluasi dan menganalisis algoritma dan sistem rekomendasi.



[1] Diakses dari https://www.journaldev.com/13639/retrofit-android-example-tutorial pada 16 April 2020
[2] Diakses dari http://surpriselib.com/ pada 17 April 2020