BAB II
TINJAUAN PUSTAKA

2.1 Universitas Komputer Indonesia

Universitas Komputer Indonesia (UNIKOM) secara resmi berdiri pada hari Selasa, tanggal 8 Agustus 2000 berdasarkan Surat Keputusan Menteri Pendidikan Nasional nomor 126/D/0/2000. UNIKOM merupakan suatu perguruan tinggi bidang teknologi, Informasi dan Komunikasi yang mempunyai tujuan sebagai pusat unggulan di bidang IPTEK. UNIKOM terus menerus melakukan peningkatan mutu, efisiensi maupun produk agar mampu bersaing antar perguruan tinggi.

2.1.1 Visi dan Misi

Misi

Menjadi Universitas terkemuka di bidang Teknologi Informasi & Komunikasi, berwawasan Global, berjiwa Entrepreneur dan menjadi Pusat Unggulan di bidang Ilmu Pengetahuan dan Teknologi yang mendukung Pembangunan Nasional serta berorientasi pada kepentingan Masyarakat, Bangsa dan Negara.

Misi

Menyelenggarakan Pendidikan Tinggi Modern berdasarkan Budaya Organisasi UNIKOM, PIQIE (*Professionalism*, *Integrity*, *Quality*, *Information Technology*, *Excellence*), dengan Sistem Pendidikan yang Kondusif dan Program–program Studi yang berbasis pada *Software*

(Perangkat Lunak), *Hardware* (Perangkat Keras), dan Entrepreneurship (Kewirausahaan) dengan mengoptimalkan Sumber Daya yang ada berdasarkan prinsip Efisiensi, Efektifitas dan Produktifitas.

Tujuan

Menghasilkan Lulusan yang unggul dibidang Teknologi Informasi & Komunikasi, Kompeten dan Handal di Bidang Studinya, berjiwa Entrepreneur, Santun dan Berbudi Luhur, Memiliki Komitmen untuk memajukan Bangsa dan Negara serta Beriman dan Bertaqwa kepada Tuhan Yang Maha Esa.

Budaya Organisasi

PIQIE (*Professionalism*, *Integrity*, *Quality*, *Information Technology*, *Excellence*)

Motto

Quality Is Our Tradition

2.1.2 Logo UNIKOM

Gambar 2.1 Logo Universitas Komputer Indonesia

Bingkai Segi Lima

Melambangkan UNIKOM berlandaskan falsafah negara yakni Pancasila dan Undang-Undang Dasar 1945.

Lingkaran Dalam Segi Lima Tempat Tulisan Berwarna Kuning

Melambangkan motto UNIKOM menuju kejayaan yakni *Quality Is Our Tradition*.

Bulatan Dalam Berwarna Biru

Melambangkan UNIKOM bertujuan menghasilkan ilmuwan unggul dan berpikiran maju yang Bertaqwa kepada Tuhan Yang Maha Esa.

Komputer

Melambangkan ciri utama UNIKOM yang memberikan pendidikan Teknologi Informasi dan Komputasi pada seluruh Jurusan yang ada di lingkungan Universitas Komputer Indonesia, menjadi Universitas Terdepan di bidang Teknologi Informasi dan Komputer serta sebagai Universitas komputer pertama di Indonesia.

Stasiun Relay

Melambangkan UNIKOM menyelenggarakan Pendidikan Tinggi kearah masyarakat industri maju dengan sistem pendidikan yang kondusif dan tenaga pengajar berkualitas untuk menghasilkan lulusan-lulusan terbaik.

Satelit

Melambangkan UNIKOM berwawasan Global dan menjadi pusat unggulan di bidang IPTEK & seni yang mendukung Pembangunan Nasional serta berorientasi pada kepentingan masyarakat, bangsa dan negara.

Cakrawala

Melambangkan indahnya menggapai Cita-cita dan mengejar ilmu setinggi Langit.

Buku

Melambangkan sumber ilmu yang tiada habis-habisnya.

2.1.3 Repositori Karya Ilmiah Unikom

Repositori merupakan salah satu bentuk akses terbuka (*open access*) terhadap karya ilmiah. Akses terbuka melalui repositori disebut cara '*green road*', dimana penulis atau lembaga mengarsipkan sendiri karya mereka untuk dapat diakses oleh publik [4]. Repositori dapat dibedakan menjadi dua yaitu repositori bidang ilmu tertentu (*subject repository*), dan repositori karya satu lembaga yang disebut repositori institusi [5]. Repositori institusi memiliki empat karakteristik yaitu jelas lembaga yang mengembangkan, kontennya ilmiah bukan populer, bersifat kumulatif yang terus bertambah setiap waktu, dan aksesnya terbuka untuk masyarakat luas [6].

Repository institusi yang dipunyai Universitas Komputer Indonesia beralamat di https://*repository*.unikom.ac.id (akses laporan penelitian). Berisikan dokumen jurnal UNIKOM, thesis S2, skripsi S1, tugas akhir D3, dan materi kuliah online.

2.2 *Similarity Document*

Konsep *similarity* sudah menjadi isu yang sangat penting di hampir setiap bidang ilmu pengetahuan [7]. Mengukur kesamaan dari dua buah dokumen merupakan bagian dari bentuk *information retrieval*, seperti *clustering*,

classification, dan beberapa bentuk lainnya [8]. Ada tiga macam teknik yang dibangun untuk menentukan nilai kemiripan dokumen diantaranya [9].

1. *Distance-based similarity measure*

   Distance-based similarity measure mengukur tingkat kesamaan dua buah objek dari segi jarak geometris dari variabel-variabel yang tercakup di dalam kedua objek tersebut. Metode *Distance-based similarity* ini meliputi *Minkowski Distance*, *Manhattan/City block distance, Euclidean distance, Jaccard distance, Dice's Coefficient*, *Cosine similarity, Levenshtein Distance, Hamming distance,* dan *Soundex distance*.

2. *Feature-based similarity measure*

   Feature-based similarity measure melakukan penghitungan tingkat kemiripan dengan merepresentasikan objek ke dalam bentuk *feature-feature* yang ingin diperbandingkan. *Feature-based similarity measure* banyak digunakan dalam melakukan pengklasifikasian atau *pattern maching* untuk gambar dan teks.

3. *Probabilistic-based similarity measure*

   Probabilistic-based similarity measure menghitung tingkat kemiripan dua objek dengan merepresentasikan dua set objek yang dibandingkan dalam bentuk *probability*. *Kullback Leibler Distance* dan *Posterior Probability* termasuk dalam metode ini.

Algoritma *Smith-Waterman* yang diimplementasikan dapat menghasilkan *local alignment* yang akurat dari perbandingan dokumen teks. *Preprocessing*

membantu performa algoritma *Smith-Waterman* saat pengisian matriks kesamaan dan pembentukan *local alignment*, *similarity* dari sistem menjadi lebih akurat namun waktu prosesnya menjadi lama apabila dibantu dengan tahapan *preprocessing* [10].

Berdasarkan penelitian A.R. Lahitani, A.E. Permanasari, dan N.A. Setiawan, dengan judul " *Cosine Similarity to Determine Similarity Measure: Study Case in Online Essay Assessment*", teknik algoritma *cosine similarity* memiliki akurasi kemiripan yang baik untuk menguji tingkat kemiripan essay [11].

Berdasarkan penelitian A. Jain, A. Jain, dan N. Chauhan, yang berjudul "*Information Retrieval* using *Cosine and Jaccard Similarity Measures in Vector Space Model*", menggunakan algoritma *cosine* dan *jaccard similarity* dengan penggabungan metode *TF-IDF* untuk memperhalus algoritma *Vector Space Model*, sehingga menghasilkan nilai yang lebih baik [12].

Berdasarkan penelitian I. Indriyanto *and* I.D. Sumitra, yang berjudul "*Measuring the Level of Plagiarism of Thesis using Vector Space Model and Cosine Similarity Methods*", dalam mengukur tingkat kemiripan dokumen menggunakan *Vector Space Model* dan *Cosine Similarity* menghasilkan nilai yang lebih baik dengan menggabungkan bersama teknik *TF-IDF* dari pada hanya dengan teknik *TF* saja [13].

Menurut M.N. Cholis, E. Yudaningtyas, dan M. Aswin hampir setiap kata khususnya dalam bahasa indonesia memiliki sinonim, sehingga perlu adanya suatu teknik yang dapat mengenali kata dengan penulisan berbeda namun

memiliki makna yang sama, karena setiap algoritma *similarity text* belum mampu membedakan kata sinonim, pada penelitiannya teknik yang digunakan adalah *Synonym recognition* dimana prosesnya dengan membandingkan kata yang ada di dalam kamus sinonim [14].

Oleh karena perubahan kata dalam kalimat akan sangat berpengaruh pada pengukuran kemiripan apabila tidak ada teknik yang dapat mendeteksi sinonim kata, dan beberapa penelitian pun dalam mengukur tingkat kemiripan dokumen jarang ada yang memakai teknik tersebut sehingga berpengaruh pada hasil kemiripan.

Hasil pengukuran dari beberapa algoritma *similarity document*, dimana dalam pengukuran ini menggunakan kalimat asli dan kalimat hasil *preprocessing*, berikut kalimat-kalimat yang akan digunakan untuk mengukur kinerja dari algoritma *similarity document*:

Tabel 2.1 Kalimat-kalimat yang digunakan untuk pengukuran

Proses	Kalimat
K1	**Kalimat asli:** *feature-based similarity measure* melakukan penghitungan tingkat kemiripan dengan merepresentasikan objek ke dalam bentuk *feature-feature* yang ingin diperbandingkan **Kalimat *Preprocessing*:** *feature-based similarity measure* melakukan penghitungan tingkat kemiripan dengan merepresentasikan objek ke dalam bentuk *feature-feature* yang ingin diperbandingkan
K2	**Kalimat asli:** repositori merupakan salah satu bentuk akses terbuka (*open access*) terhadap karya ilmiah **Kalimat *Preprocessing*:** repositori rupa salah satu bentuk akses buka *open access* hadap karya ilmiah
K3	**Kalimat asli:**

Tabel 2.1 Kalimat-kalimat yang digunakan untuk pengukuran (Lanjutan)

Proses	Kalimat
	mengukur kesamaan dari dua buah dokumen merupakan bagian dari bentuk *information retrieval*, seperti *clustering*, *classification*, dan beberapa bentuk lainnya **Kalimat *Preprocessing*:** ukur sama dari dua buah dokumen rupa bagi dari bentuk *information retrieval* seperti *clustering classification* beberapa bentuk lain
K4	**Kalimat asli:** dalam kasus ini penulis akan mencoba menggabungkan teknik *text summarization* dan *similarity document* **Kalimat *Preprocessing*:** dalam kasus ini tulis akan coba gabung teknik *text summarization similarity document*
K5	**Kalimat asli:** kompleksitas dapat dibagi menjadi dua jenis kompleksitas yaitu *time complexity* **Kalimat *Preprocessing*:** kompleksitas dapat bagi menja dua jenis kompleksitas yaitu *time complexity*

Tabel 2.2 Pengukuran algoritma *similarity document*

Algoritma	Kemiripan					Rata-rata
	K1	K2	K3	K4	K5	
Smith-Waterman Similarity	87.79%	91.33%	93.08%	88.10%	97.30%	91.64%
Cosine Similarity	62.62%	75.00%	75.57%	72.06%	83.33%	73.72%
Jaccard Similarity	74.09%	40.95%	65.65%	47.14%	39.00%	53.36%

Dari hasil tabel pengukuran di atas, dapat disimpulkan bahwa algoritma *Smith-Waterman* memiliki hasil yang cukup baik dengan nilai rata-rata sebesar 91,64%. Maka penulis dalam penelitian ini menggunakan algoritma *Smith-Waterman* dalam mengecek kemiripan suatu kalimat.

2.3 *Text Summarization*

Peringkasan teks didefinisikan sebagai proses pemadatan teks menjadi versi yang lebih pendek namun tetap mengandung informasi yang bisa mencerminkan keseluruhan teks [15]. proses mengurangi dokumen teks dengan program komputer untuk menciptakan sebuah ringkasan yang mempertahankan poin yang paling penting dari dokumen adalah dengan Metode Ekstraksi dan metode abstraksi. Menurut Hovy, ringkasan adalah teks yang dihasilkan dari sebuah teks atau banyak teks, yang mengandung isi informasi dari teks asli dan panjangnya tidak lebih dari setengah teks aslinya [16].

Terdapat dua pendekatan pada peringkasan teks, yaitu ekstraksi (*shallower approaches*) dan abstraksi (*deeper approaches*). Pada teknik ekstraksi, sistem menyalin unit-unit teks yang dianggap paling penting atau paling informatif dari teks sumber menjadi ringkasan. Unit-unit teks yang disalin dapat berupa klausa utama, kalimat utama, atau paragraf utama. Sedangkan teknik abstraksi melibatkan parafrase dari teks sumber. Teknik abstraksi mengambil intisari dari teks sumber, kemudian membuat ringkasan dengan menciptakan kalimat-kalimat baru yang merepresentasikan intisari teks sumber dalam bentuk berbeda dengan kalimat-kalimat pada teks sumber. Pada umumnya, abstraksi dapat meringkas teks lebih kuat daripada ekstraksi, tetapi sistemnya lebih sulit dikembangkan karena mengaplikasikan teknologi *natural language generation* yang merupakan bahasan yang dikembangkan tersendiri.

Teknik *Text Summarization* terdapat 3 metode secara umum yaitu :

a. *Extraction-based summarization*

Dua jenis *summarization* sering dibahas dalam literatur adalah ekstraksi *keyphrase*, di mana tujuannya adalah untuk memilih kata-kata individu atau frase untuk "*tag*" sebuah dokumen, dan *summarization* dokumen, di mana tujuannya adalah untuk memilih seluruh kalimat untuk membuat ringkasan paragraf pendek.

b. *Abstraction-based summarization*

Teknik ekstraksi hanya menyalin informasi yang dianggap paling penting oleh sistem untuk ringkasan (misalnya, klausa kunci, kalimat atau paragraf), sedangkan abstraksi melibatkan parafrase bagian dari dokumen sumber. Secara umum, abstraksi dapat menyingkat teks lebih kuat dari ekstraksi, tetapi program yang bisa melakukan hal ini lebih sulit untuk mengembangkan karena mereka memerlukan penggunaan teknologi *natural language generation*.

c. *Maximum entropy-based summarization*

Meskipun automating *abstractive summarization* adalah tujuan dari penelitian *summarization*, sistem yang paling praktis didasarkan pada beberapa bentuk adalah *summarization* ekstraktif. *Maximum entropy-based summarization* telah berhasil diterapkan.

Menurut Mosa, M. A., Hamouda, A., & Marei, dalam penelitiannya yang berjudul "*Ant Colony Heuristic for User-Contributed Comments Summarization*" menunjukan hasil pengukuran akurasi dari berbagai algoritma yang ada, bahwa

untuk algoritma *Ant Colony Optimization* memiliki kinerja lebih baik dari algoritma yang lain dalam penelitiannya, Berikut tabel pengukuran hasil perbandingan algoritma *summarization* [17].

Tabel 2.3 Perbandingan algoritma *summarization* [17].

Algorithm	Precession	Recall	*F-measure*
ACO	**95.1%**	**89.3%**	**92.1%**
LexRank	84.2%	77.9%	80.9%
PageRank	81.2%	76.5%	78.8%
Mead	78.8%	70.3%	74.3%
TFIDF	73.3%	60.8%	66.5%
MI	66.0%	60.1%	62.9%
LexRank & K-means	85.6%	79.1%	82.2%
PageRank & K-means	85.0%	78.0%	81.3%
Mead & K-means	82.7%	77.6%	80.0%
TFIDF & K-means	79.0%	73.5%	76.2%
MI & K-means	82.2%	75.6%	78.8%
LexRank & Hierarchical	88.5%	81.3%	84.7%

Dari tabel diatas algoritma *Ant Colony Optimization* memiliki akurasi dan kinerja lebih baik dari algoritma yang lain dengan nilai *f-measure* sebesar 92.1%, maka penulis dalam penelitian akan menggunakan algoritma *Ant Colony Optimization* dalam optimalisasi hasil ringkasan *text*.

2.4 *Preprocessing Text*

Berdasarkan ketidak teraturan struktur data teks, maka proses sistem temu kembali informasi ataupun *text mining* memerlukan beberapa tahap awal yang pada intinya adalah mempersiapkan agar teks dapat diubah menjadi lebih terstruktur, Salah satu implementasi dari *text* mining adalah tahap *Text Preprocessing* [18]. Tahap *Text Preprocessing* adalah tahapan dimana aplikasi

melakukan seleksi data yang akan diproses pada setiap dokumen. Proses *preprocessing* ini meliputi :

2.4.1 Case Folding

Tidak semua dokumen teks konsisten dalam penggunaan huruf kapital. Oleh karena itu, peran *Case Folding* dibutuhkan dalam mengkonversi keseluruhan teks dalam dokumen menjadi suatu bentuk standar (biasanya huruf kecil atau *lowercase*).

2.4.2 Tokenizing

Tahap *Tokenizing* adalah tahap pemotongan *string input* berdasarkan tiap kata yang menyusunnya. Contoh dari tahap ini dapat dilihat dibawah ini.

Tabel 2.4 Hasil Proses Tokenizing

Text Input	**Hasil *Tokenizing***
metode yang digunakan dalam pendeteksian kemiripan	metode yang digunakan dalam pendeteksian kemiripan

2.4.3 *Filtering*

Tahap *Filtering* adalah tahap mengambil kata-kata penting dari hasil token. dimana kata-kata yang tidak deskriptif dihilangkan. Contoh “yang”, “dan”, “di”, “dari” dan seterusnya. Contoh dari tahap ini sebagai berikut:

Tabel 2.5 Hasil Proses Filtering

Sebelum *Filtering*	**Setelah *Filtering***
metode yang digunakan	metode digunakan dalam

Tabel 2.5 Hasil Proses Filtering (Lanjutan)

Sebelum *Filtering*	**Setelah *Filtering***
dalam pendeteksian kemiripan	pendeteksian kemiripan

2.4.4 *Stemming*

Proses *Stemming* pada teks berbahasa Indonesia berbeda dengan *Stemming* pada teks berbahasa Inggris. Pada teks berbahasa Inggris, proses yang diperlukan hanya proses menghilangkan sufiks. Sedangkan pada teks berbahasa Indonesia semua kata imbuhan baik itu sufiks dan prefiks juga dihilangkan.

Berdasarkan penelitian yang telah dilakukan oleh Gede Aditra Pradnyana, I Komang Ari Mogi, memperoleh hasil bahwa keakuratan dari Algoritma *Porter* tidak memiliki perbedaan yang signifikan terhadap Algoritma *Nazief & Adriani*, namun dari segi waktu yang dibutuhkan masing-masing algoritma dalam melakukan prosesnya (*stemming*), Algoritma *Porter* lebih cepat dibandingkan dengan Algoritma *Nazief & Adriani* [19].

Dalam penelitian ini penulis menggunakan algoritma *Porter* dalam melakukan proses *stemming*,

Berikut langkah-langkah algoritma *Porter* [20]:

1. Hapus *Particle*.
2. Hapus *Possesive Pronoun*.
3. Hapus awalan pertama. Jika tidak ada lanjutkan ke langkah 4a, jika ada cari maka lanjutkan ke langkah 4b.
4. a. Hapus awalan kedua, lanjutkan ke langkah 5a.

b. Hapus akhiran, jika tidak ditemukan maka kata tersebut diasumsikan sebagai *root word*. Jika ditemukan maka lanjutkan ke langkah 5b.

5. a. Hapus akhiran. Kemudian kata akhir diasumsikan sebagai *root word*.

   b. Hapus awalan kedua. Kemudian kata akhir diasumsikan sebagai *root word*.

contoh hasil *Stemming* menggunakan algoritma *Porter*:

Tabel 2.6 Hasil Proses Stemming

Sebelum *Stemming*	**Hasil *Stemming***
metode digunakan dalam pendeteksian kemiripan	metode guna dalam deteksi mirip

2.4.5 Stopword *Removal*

Sebagian besar bahasa resmi memiliki kata yang hampir selalu muncul pada dokumen-dokumen teks. Kata-kata ini umumnya tidak memiliki arti lebih dalam mencari informasi. Sebuah sistem *Text Retrieval* biasanya disertai dengan sebuah *Stoplist*. *Stoplist* berisi sekumpulan kata yang 'tidak relevan', namun sering sekali muncul dalam sebuah dokumen [21].

2.5 *Term Frequency And Inverse Document Frequency (TF-IDF)*

Metode ini merupakan metode untuk menghitung bobot setiap setiap kata yang paling umum digunakan dalam multi disiplin ilmu *natural language* processing [22]. Dengan menggunakan *TF IDF* ini dapat diketahui bobot dari

setiap kata/*term*. *TF IDF* ini merupakan suatu algoritma yang paling umum digunakan dalam *information retrieval*.

2.5.1 *Term Frequency* (*TF*)

Term Frequency digunakan untuk mengukur banyaknya kemunculan kata dari dalam suatu dokumen [23]. Contoh pada dokumen "D1" terdapat 1000 kata, dan kata "algoritma" terdapat 15 kata dalam dokumen tersebut, maka perhitungan *term frequency* sebagai berikut:

TF = 15/1000 = 0,015

2.5.2 *Inverse Document Frequency* (*IDF*)

Metode *IDF* merupakan sebuah perhitungan dari bagaimana *term* didistribusikan secara luas pada koleksi dokumen yang bersangkutan, berbeda dengan *TF* yang semakin sering frekuensi kata muncul maka nilai semakin besar, dalam *IDF*, semakin sedikit frekuensi kata muncul dalam dokumen, maka makin besar nilainya [24]. Untuk menentukan besaran nilai *IDF*, maka menggunakan rumus berikut:

$$IDF = \log\left(D/df_j\right) \quad \ldots\ldots\ldots\ldots\ldots\ldots (1)$$

Keterangan :

D = jumlah semua dokumen dalam koleksi

df_j = jumlah dokumen yang mengandung *term* (tj)

2.5.3 *Term Frequency - Inverse Document Frequency* (*TF-IDF*)

Rumus umum untuk *Term Weighting TF-IDF* adalah penggabungan dari perhitungan *TF* dengan *IDF* dengan cara mengalikan nilai *TF* dengan nilai *IDF* [25]:

$$W_{ij} = tf_{ij} x idf_j \quad \text{..... (2)}$$

Keterangan:

W_{ij} = bobot *term* (t_j) terhadap dokumen

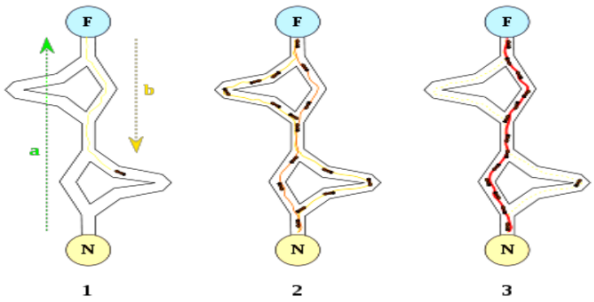
tf_{ij} =jumlah kemunculan *term* (t_j) dalam dokumen

d_{fj} = jumlah dokumen yang mengandung *term* (t_j)

2.6 *Ant Colony Optimization Algorithm*

Ant Colony Optimization merupakan pengembangan dari *Ant Colony*. Secara informal, *Ant Colony Optimization* bekerja sebagai berikut: pertama kali, sejumlah m semut ditempatkan pada sejumlah n titik berdasarkan beberapa aturan inisialisasi (misalnya, secara acak). Setiap semut membuat sebuah *tour* (yaitu, sebuah solusi jalur evakuasi yang mungkin) dengan menerapkan sebuah aturan transisi status secara berulang kali. Selagi membangun tournya, setiap semut juga memodifikasi jumlah *pheromone* pada *edge-edge* yang dikunjunginya dengan menerapkan aturan pembaruan *pheromone local* yang telah disebutkan tadi [26]. Menurut Deng Wu, Junjie Xu, *and* Huimin Zhao Algoritma *Ant Colony Optimization* terdiri dari sejumlah iterasi, Di setiap iterasi, sejumlah semut membangun sebuah solusi dengan menggunakan heuristik, solusi-solusi ini

diwakili oleh sebuah jejak feromon [27]. Berikut langkah-langkah Penyelesaian Komputasi Pada *Ant Colony Optimization* [28]:



Gambar 2.2 Langkah-langkah komputasi ACO

Kemampuan individual terbatas semut telah mampu menemukan jalan terpendek antara sumber makanan dan sarang.

1. Semut pertama menemukan sumber makanan (F), melalui cara apapun (a), kemudian kembali ke sarang (N), meninggalkan jejak (b).
2. Semut berikutnya mengikuti empat cara yang mungkin, tetapi iya memilih jalur sebagai rute terpendek.
3. Semut mengambil rute terpendek, dan jejak route yang panjang akan hilang.

Seekor semut k ketika melewati ruas akan meninggalkan *pheromone*. Jumlah *pheromone* yang terdapat pada ruas ij setelah dilewati semut k diberikan dengan rumus:

$$\tau_{i,j} \leftarrow \tau_{i,j} + \Delta\tau^{k} \quad \text{..... (3)}$$

Dengan meningkatnya nilai *pheromone* pada ruas i-j, maka kemungkinan ruas ini akan dipilih lagi pada iterasi berikutnya semakin besar. Setelah sejumlah

simpul dilewati maka akan terjadi penguapan *pheromone* dengan aturan sebagai berikut:

$$\tau_{i,j} \leftarrow (1-\rho)\tau_{i,j}, j; \forall (i,j) \in A \text{ .....(4)}$$

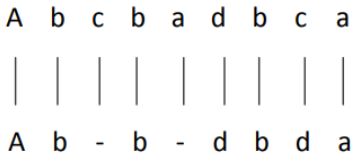
Penurunan jumlah *pheromone* memungkinkan semut untuk mengekplorasi lintasan yang berbeda selama proses pencarian. Ini juga akan menghilangkan kemungkinan memilih lintasan yang kurang bagus. Selain itu, ini juga membantu membatasi nilai maksimum yang dicapai oleh suatu lintasan *pheromone*.

2.7 *Smith Waterman Algorithm*

Algoritma *Smith-Waterman* diperkenalkan oleh Temple F. *Smith* dan Michael S. *Waterman* pada tahun 1981 di sebuah Jurnal Molekular Biologi dalam papernya yang berjudul "*Identification of Common Molecular Subsequences"* [29]. Algoritma ini digunakan untuk mencari kesamaan yang paling signifikan (*local alignment*) dari dua buah rangkaian sekuens gen, dimana setiap sekuens gen dalam rangkaian yang satu dibandingkan dengan sekuens gen rangkaian yang lain berdasarkan kesamaan strukturnya. Signifikan dalam hal ini memenuhi suatu nilai ambang batas (*threshold*) tertentu. Cara kerja algoritma ini yaitu dengan cara membandingkan dua buah *string* dengan mengidentifikasikan apakah terdapat bagian-bagian yang sama di antara kedua *string* tersebut.

Algoritma *smith waterman* merupakan algoritma klasik yang telah dikenal luas dalam bidang bioinformatika sebagai metode yang dapat mengidentifikasi *local similarities* (penjajaran sekuens) yaitu proses penyusunan dua *local sequences* (rangkaian atau susunan) neucleotide atau protein *sequences* sehingga

kemiripan antara dua *sequences* tersebut akan terlihat. Berikut ini gambar optimal *aligment* dari dua *substring* dapat dilihat pada gambar dibawah ini:



Gambar 2.1 *Optimal Aligment* Dari Dua *Substring*

Algoritma standar *Smith-Waterman* yang dipakai untuk perhitungan *local alignment* berdasarkan dokumen dari situs Baylor College of Medicine HGSC adalah [30]:

1. Menambahkan sebuah nilai pada setiap nilai perbandingan
    a. Menggunakan nilai positif, apabila memiliki kemiripan.
    b. Menggunakan nilai negatif, apabila memiliki perbedaan
2. Inisialisasi awal matriks dengan nilai 0 (nol).
3. Semua nilai yang terdapat dalam matriks apabila lebih kecil dari 0 (nol), maka nilai dianggap 0 (nol).
4. Memulai *traceback* dari nilai yang tertinggi yang ditemukan dimanapun pada matrik.
5. Perhitungan dilanjutkan hingga skor bernilai 0 (nol).

Algoritma ini juga dapat digunakan dalam permasalahan *string matching* dengan menerapkan *dynamic programming*, yaitu metode penyelesaian masalah dari suatu permasalahan dengan menguraikan solusi menjadi sekumpulan langkah-langkah sedemikian sehingga solusi dari sebuah persoalan dapat dipandang dari serangkaian keputusan yang saling berkaitan.

Rumusan skema pemrograman dinamis algoritma *Smith-Waterman* menjadi dua bagian, yaitu:

1. Didefinisikan Sij menjadi nilai maksimum yang didapat dari proses perbandingan sebuah *string* A pada posisi ke-i dengan sebuah *substring* B pada posisi ke-j. Hubungan rekurens standar untuk Sij yaitu :
   a. Jika A(i) = B(j) maka Sij = Si-1j-1+h, atau
   b. Jika A(1) ≠ B(j) maka Sij = max(0,Si-1,j-d, Si,j-1-d, Si-1,j-1-r)
   c. Dimana kondisi awal adalah Si,0 = S0,j = 0 untuk semua i,j.
2. Digunakan ide *traceback* path untuk mengkonstruksikan sebuah *local aligment* yang optimal pada posisi ke-i *substring* A dan posisi *substring* ke-j *substring* B agar lebih jelas terlihat. Dengan diberikan sel (i,j) dapat didefinisikan sebuah sel *parent* sebagai berikut :
   a. Jika Sij = 0 maka sel (i,j) tidak mempunyai *parent*.
   b. Jika A(i) = B(j), maka sel (i,j) mempunyai *parent* sel (i-1,j-1).
   c. Sebagai tambahan, sel (i,j) mempunyai parent yaitu untuk sel (p,q){(i-1,j),(i,j1)} sehingga Sij = Spq-d, dan atau sel (i-1,j-1) jika Sij = Si-1,j-1-r.

   Jadi, untuk setiap sel yang mengandung nilai tidak nol akan mempunyai sedikitnya satu *parent*, dan mungkin dapat memiliki tiga buah *parent*.

2.8 *Root Mean Square Error (RMSE)*

Cara yang cukup sering digunakan dalam mengevaluasi suatu metode yaitu menggunakan metode *Root Mean Squared Error* (*RMSE*). *Root Mean Squared*

Error (*RMSE*) digunakan untuk mengetahui nilai *error* dari nilai yang direkomendasikan oleh sistem terhadap nilai sebenarnya [14].

$$RMSE = \sqrt{\frac{\sum_{i=1}^{n}(A-F)^2}{n}} \quad \text{.....(5)}$$

Dimana:

A = Nilai observasi (dari hasil kemiripan dokumen *brute force*)

F = Nilai prediksi (dari hasil penggabungan dua teknik)

n = banyaknya data

Keakuratan metode estimasi kesalahan pengukuran diindikasikan dengan adanya *RMSE* yang kecil. Metode estimasi yang mempunyai *RMSE* lebih kecil dikatakan lebih akurat daripada metode estimasi yang mempunyai *RMSE* lebih besar [31].

2.9 *Big O Notation*

Kompleksitas suatu algoritma umumnya dihitung menggunakan notasi *Big-O*. Kompleksitas dapat dibagi menjadi dua jenis kompleksitas yaitu *Time Complexity*, Menyatakan berapa lama suatu algoritma berjalan ketika *runtime* berdasarkan *input* yang diberikan. Biasa dinotasikan dengan *Big-O notation*, dan *Space Complexity*, Menyatakan berapa banyak ruang dalam memori yang dibutuhkan dalam suatu algoritma ketika beroperasi [32]. Dalam penelitian ini penulis hanya akan menggunakan *Time Complexity* dalam mengukur komputasi dari suatu model yang diusulkan.

Dalam menghitung *time complexity* (yang biasanya dinotasikan dengan *Big-O notation*), ada beberapa aturan yang perlu dilakukan disini [33]:

1. Abaikan konstanta, misalkan O(N + 2), maka dianggap O(N) saja.
2. Abaikan non dominant terms,misalkan $O(N^2 + N)$, maka dianggap $O(N^2)$ saja.

Beberapa contoh *time complexity* yang menyatakan berapa lama suatu algoritma berjalan ketika *runtime* berdasarkan *input* yang diberikan.

```
int add(int a, int b) {
    return a + b;
}
```

Fungsi diatas memiliki *time complexity* O(1) dikarenakan ia hanya menjalankan sekali instruksi return, berapapun *input* yang dimasukkan kedalam fungsi.

```
double average(double[] numbers) {
    double sum = 0;
    for(double number: numbers) {
        sum += number;
    }
    return sum / numbers.length;
}
```

Fungsi diatas memiliki *time complexity* O(n) dikarenakan ia akan menjalankan *looping* untuk menjumlahkan bilangan-bilangan yang ada didalam array. Jumlah literasinya bergantung pada panjang array yang dimasukkan kedalam fungsi.

```
int func(int n) {
    int count = 0;
    for (int i = 1 ; i <= n ; i++) {
        for (int j = 1 ; j <= i ; j++) {
            count++;
        }
    }
    return count;
}
```

Fungsi diatas memiliki *time complexity* $O(n^2)$ dikarenakan ia akan menjalankan *looping* sebanyak masukan dari nilai n kedalam fungsi.