OPEN ACCESS
Indo-JC
Indonesia Journal of Computing
ISSN 2460-9056
socj.telkomuniversity.ac.id/indojc

IND. JOURNAL ON COMPUTING
VOL. 4, ISSUE. 2, SEPT 2019. PP. 177-188
DOI:10.21108/INDOJC.2019.4.2.342

Sistem Pencarian Lintas Ayat Alquran Berdasarkan Kesamaan Fonetis

Eki Rifaldi [#1], Moch Arif Bijaksana [#2], Kemas Muslim Lhaksamana [#3]

[#] *Fakultas Infromatika, Universitas Telkom*
Jl. Telekomunikasi No. 1, Terusan Buah Batu, Bandung, Indonesia, 40257

[1] rifaldieki@student.telkomuniversity.ac.id

[2] arifbijaksana@telkomuniversity.ac.id

[3] kemasmuslim@telkomuniversity.ac.id

Abstract

Finding Arabic texts in the Quran that have many verses and differences in language with Indonesian raises its own difficulties for verse searching by Muslim communities. A phonetic-based verse-seeking system is needed which allows the user to search for verses using Latin aphabet that represents the user's pronunciation sound. For example, if searched for the word الْحَمْدُ لِلَّهِ then the system will display all the verses that have similar sounds with the keywords. For now, there is already a Quran verse search system using phonetic string matching, but limited to finding only one verses based on query. Then if searched for words across verses يَوْمِ الدِّيْنِ (٤) إِيَّاكَ by matching the string in the database, The system cannot provide the results of the search for two verses at once. Cross-paragraph search is needed because users don't necessarily know whether *query* is used in one verse or more. Therefore, a Quran verse search system was built based on phonetic similarities which can pass verses. The N-gram algorithm in the form of trigram is used to find verses that have phonetic similarities because they have high Mean Average Precision (MAP) for long keywords. To search across verses, the next five verse's trigrams are added to the end of the current verse's trigram. And as the results, MAP value is 0.9 and Recall is 0.93.

Keywords: quran, search system, phonetic similarity, string matching, n-gram, cross verse.

Abstrak

Mencari teks Arab dalam Alquran yang memiliki banyak ayat dan perbedaan bahasa dengan Bahasa Indonesia menimbulkan kesulitan tersendiri untuk pencarian ayat oleh masyarakat muslim. Dibutuhkan sistem pencarian ayat Alquran berbasis fonetis yang dapat memudahkan pengguna dalam mencari ayat menggunakan tulisan Latin berhuruf alfabet yang merepresentasikan bunyi pengucapan pengguna. Sebagai contoh, jika dilakukan pencarian kata الْحَمْدُ لِلَّهِ maka sistem akan menampilkan seluruh ayat yang memiliki kemiripan bunyi dengan kata kunci. Untuk saat ini, sudah ada sistem pencarian ayat Alquran dengan menggunakan *phonetic string matching*, namun terbatas hanya dapat menemukan ayat berdasarkan *query* yang tidak lintas ayat. Kemudian jika dilakukan pencarian *query* lintas ayat يَوْمِ الدِّيْنِ (٤) إِيَّاكَ dengan pencocokan *string* dalam *database*, maka sistem tidak dapat memberikan hasil pencarian dua ayat sekaligus. Pencarian lintas ayat sangat dibutuhkan karena pengguna belum tentu tahu apakah *query* yang digunakan itu dalam satu ayat atau lebih. Oleh karena itu, dibangun suatu sistem pencarian ayat Alquran berdasarkan kemiripan bunyi (fonetis) yang dapat melintasi ayat. Algoritma N-gram berupa trigram digunakan untuk menemukan ayat-ayat yang memiliki kemiripan bunyi (fonetis) karena memiliki *Mean Average Precision* (MAP) yang tinggi untuk kata kunci panjang. Untuk mencari lintas ayat, lima buah trigram ayat selanjutnya ditambahkan ke ujung trigram ayat sebelumnya. Kemudian diperoleh nilai MAP 0,9 dan Recall 0,93.

Kata Kunci: alquran, sistem pencarian, kemiripan fonetis, *string matching*, n-gram, lintas ayat.

RECEIVED ON AUGUST 2019. ACCEPTED ON SEPT 2019

I. Pendahuluan

AL-QURAN adalah Firman Allah S.W.T. yang diturunkan kepada Nabi Muhammad SAW melalui Malaikat Jibril sebagai mukjizat dan berfungsi sebagai hidayah (petunjuk).

Banyaknya ayat di dalam Alquran menimbulkan kesulitan tersendiri dalam pencarian ayat/kalimat/kata di dalam Alquran. Bahasa yang digunakan di Alquran adalah bahasa Arab, sehingga cukup menyulitkan masyarakat muslim Indonesia untuk melakukan pencarian ayat berdasarkan kata karena perbedaan bahasa dan jenis huruf yang digunakan.

Penelitian tentang pengembangan sistem untuk membantu pencarian ayat Alquran telah dilakukan sejak lama. Saat ini juga telah banyak dikembangkan aplikasi perangkat lunak untuk mencari ayat Alquran. Diantaranya yaitu penerapan algoritma N-gram dalam sistem pencarian Lafzi [1] menghasilkan MAP 87,71% dan Recall 95,61%, kombinasi Algoritma Soundex dan Cosine Similarity [2] menghasilkan MAP 81,81% dan Recall 80,81%, kombinasi Algoritma Soundex dan Damerau-Lavenstein [3] menghasilkan MAP 77,89% dan Recall 90,72%, kombinasi Algoritma Metaphone dan Levenstein Distance menghasilkan [4] MAP 79,00% dan Recall 82,46%, kombinasi Algoritma Double Metaphone dan Jaro Winkler [5] menghasilkan MAP 84,36% dan Recall 92,81%, Sistem Pencarian Islamic City [2] menghasilkan MAP 71,94% dan Recall 87,37%. Di samping hasil MAP dan Recall yang berbeda-beda dari masing-masing algoritma/sistem tersebut, belum ada yang dapat memfasilitasi pencarian ayat Alquran berdasarkan kata kunci yang melintasi ayat. Jika dilakukan pencarian akhir ayat pertama dan awal ayat kedua secara manual, maka akan memberikan hasil yang sangat banyak dan membingungkan.

Umumnya sistem pencarian ayat Alquran berdasarkan bunyi digunakan saat seseorang telah mendengar bunyi Ayat tertentu kemudian penasaran dan ingin mencarinya. Maka kata kunci (*query*) yang dimasukkan adalah potongan ayat Alquran yang terlintas di pikiran orang tersebut tanpa mengetahui atau mempertimbangkan apakah kata tersebut berada di awal, akhir atau melintasi dua ayat. Hal tersebut menjadi alasan kenapa pencarian ayat Alquran lintas ayat sangat penting.

Oleh karena itu, dibutuhkan suatu sistem pencarian ayat Alquran yang dapat memfasilitasi masyarakat muslim dalam mencari ayat-ayat Alquran berdasarkan kemiripan bunyi yang dapat melintasi ayat dan tanda baca, serta sesuai dengan pelafalan masyarakat Indonesia. Hal tersebut dapat dilakukan dengan menggunakan algoritma N-gram yang dimanipulasi.

II. Studi Terkait

A. Sistem Pencarian Lafzi

Lafzi merupakan sistem pencarian ayat Al-Quran berbasis kemiripan fonetis yang lebih sesuai dengan representasi pelafalan orang Indonesia. Lafzi menggunakan metode pengodean fonetis yang didasarkan pada pemadanan aksara Arab-Latin yang digunakan di Indonesia serta kemiripan cara pelafalan huruf-huruf dalam Al-Quran. Metode pencarian yang digunakan adalah pencarian dengan trigram yang diterapkan pada kode fonetis dengan ukuran kesamaan yang ditentukan. Kinerja sistem paling baik pada pencarian dengan menggunakan kata kunci vokal, menggunakan peringkat jumlah, serta untuk pencarian lafal tanpa memperhatikan makna. Sedangkan pemeringkatan berdasarkan posisi kemunculan hanya dapat memperbaiki *precission* pada titik *recall* rendah [1].

B. Pencocokan String berdasarkan kemiripan bunyi (Phonetic String Matching)

Pencocokan *string* berdasarkan kemiripan bunyi (*phonetic string matching*) merupakan pencocokan *string* dengan dasar kemiripan dari segi pengucapannya meskipun ada perbedaan penulisan dua *string* yang dibandingkan tersebut. Dan beberapa algoritma *phonetic string matching* antara lain adalah Soundex, Metaphone, Caverphone, Phonex, NYSI-IS, Jaro Winkler dan lain-lain [6].

1) Algoritma Soundex, pertama kali dipatenkan oleh Margaret O'Dell and Robert C. Russell pada tahun 1918, tetapi algoritma ini kemudian disempurnakan lagi. Algoritma Soundex menghasilkan kode fonetik dengan panjang empat karakter untuk semua panjang *string* masukan. Algoritma Soundex akan menghasilkan empat buah kode fonetis dengan aturan tertentu. Algoritma ini memiliki

kelebihan dapat mencari frasa berdasarkan cara pengucapannya. Namun, algoritma ini juga memiliki beberapa kekurangan seperti urutan hasil pencariannya akan acak serta bergantung pada inisial kata, selain itu konsonan yang tidak berbunyi juga berpengaruh pada hasil pencarian.

2) Algoritma Metaphone, merupakan algoritma yang melakukan penanganan secara khusus terhadap setiap fonem (satuan bunyi bahasa) dalam kata.
3) Algoritma Double Metaphone, merupakan algoritma penyempuraan dari Metaphone dan algoritma terdahulu, yang ditulis oleh Lawrence Philips. Ditulis *double* karena algoritma ini mengembalikan primary dan secondary code untuk sebuah *string*.
4) Algoritma Caverphone, merupakan algoritma yang masih baru dan berusaha menyempurnakan algoritma-algoritma terdahulu.

C. Algoritma N-Gram

N-gram merupakan salah satu proses yang secara luas digunakan dalam penambangan teks. (*text mining*) dan pengolahan bahasa. Secara N-gram merupakan sekumpulan kata yang diberikan dalam sebuah paragraf dan ketika menghitung N-gram biasanya dilakukan dengan menggerakkan satu kata ke depan (meskipun dalam prosesnya terdapat suatu proses dimana kata yang dimajukan sejumlah X kata). Dengan kata lain, N-gram adalah potongan N-karakter yang diambilkan dari suatu *string*. Untuk mendapatkan N-gram yang utuh ditempuh dengan menambahkan *blank* pada awal dan akhir *string*. Misalnya suatu *string* "TEXT" setelah ditambah aal dan akhir dengan "_" sebagai pengganti blank akan didapat N-gram sebagai berikut [7]:

- Unigram : T,E,X,T
- Bigram : _T, TE, EX, XT, dan T
- Trigram : _TE,TEX,EXT, XT_ dan T_ _
- Quadgram : _TEX, TEXT, EXT_, EX_ _, X_ _ _

Dapat disimpulkan bahwa untuk string berukuran n akan dimiliki n unigram dan n+1 bigram, n+1 trigram, n+1 quadgram dan seterusnya. Penggunaan N-gram untuk matching kata memiliki keuntungan sehingga dapat diterapkan pada *recovery* pada input karakter ASCII yang terkena noise, interpretasi kode pos, *information retrieval* dan berbagai aplikasi dalam pemrosesan bahasa alami. Keuntungan N-gram dalam *string matching* adalah berdasarkan karakteristik N-gram sebagai bagian dari suatu string, sehingga kesalahan pada sebagian string hanya akan berakibat perbedaan pada sebagian Ngram. Sebagai contoh jika N-gram dari dua *string* dibandingkan, kemudian kita menghitung cacah N-gram yang sama dari dua *string* tersebut maka akan didapatkan nilai similaritas atau kemiripan dua *string* tersebut yang bersifat resistan terhadap kesalahan tekstual [8]. Kemiripan antara kata SUMA dengan SAMA (ada perbedaan 1 huruf), dapat diukur derajat kesamaan dengan cara menghitung berapa buah N-gram yang diambil dari dua kata tersebut yang bernilai sama, yaitu:

- SUMA: _S, SU, UM, MA, A_ , SAMA : _S, SA, AM, MA, A_ kesamaan :3

Sementara antara kata SUMA dengan SUWI (ada perbedaan 2 huruf), nilai kesamaan adalah:

- SUMA: _S, SU, UM, MA, A_ , SUWI : _S, SU, UW, WI,I_ kesamaan : 2

Sehingga dapat disimpulkan bahwa kemiripan atau kesamaan antara SUMA-SAMA lebih dari pada SUMA-SUWI.

D. Longest Common Subsequence (LCS)

Longest Common Subsequences (LCS) adalah masalah mencari uparangkaian bersama (*common subsequence*) terpanjang dari beberapa (biasanya hanya 2) buah rangkaian. Uparangkaian (*subsequence*) dari sebuah string S adalah sekumpulan karakter yang ada pada S yang urutan kemunculannya sama. LCS digunakan untuk mencari sebuah kemiripan antara *query* pencarian dan data yang data yang ada dalam *database*. LCS tidak hanya berfungsi untuk menemukan masing-masing karakter yang sama antara *query* pencarian dan *database* tetapi mencari karakter yang sama dan juga karakter ysng berurutan. LCS akan menemukan kesamaan dengan cara mencocokkan satu per satu setiap karakter *query* pencarian dengan *query* dalam *database* sampai karakter *query* pencarian memiliki kesamaan seluruhnya ataupun kesamaannya hanya sebagian. Ketika LCS menemukan data yang memiliki kesamaan seluruh karakter *query* pencarian maka data tersebut adalah hasil paling sempurna dalam sebuah pencarian dengan LCS.

Namun ketika LCS tidak menemukan semua karakter antara *query* pencarian dan *database* sama atau mirip keseluruhannya maka hasil terbaik adalah data yang kesamaannya dimulai dari awal sampai akhir kesamaan paling banyak diantara data-data yang lain sehingga ketika sebuah pencarian yang beberapa karakternya tidak ditemukan maka LCS akan mengabaikan karakter sisa yang tidak terbaca oleh LCS. Perhatikan juga bahwa secara umum, satu *query* dapat memiliki lebih dari satu LCS. Misalnya, "abd" dan "acd" adalah LCS dari "abcd" dan "acbd" [9].

Contoh penggunaan LCS yaitu jika ada permintaan pencarian "*onetwothreefourfive*" maka *query* dalam *database* akan dicari kesamaannya dengan cara:

1) "*onetwothree*" akan menjadi hasil atau solusi terbaik karena ada 11 karakter yang sama dan berurutan dari 19 karakter yang dicari.
2) "*onetwofour*" dalam kasus pencarian yang sama, hasil pencarian sudah benar namun bukan merupakan hasil yang terbaik karena hanya terdapat 6 karakter yang sama dan berurutan.

E. Jaro Winkler Distance

Algoritma Jaro Winkler merupakan salah satu algoritma dalam metode *Approximate String Matching* yaitu metode yang digunakan untuk mengukur kesamaan antar string [10]. Algoritma Jaro Winkler Distance merupakan varian dari Jaro Distance *metric* yang biasanya digunakan di dalam pendeteksian duplikat. Semakin tinggi Jaro Winkler Distance untuk dua string maka semakin mirip dengan string tersebut. Nilai normalnya ialah 0 menandakan tidak ada kesamaan dan lebih dari 0 yang menandakan adanya kesamaan. Berdasarkan penelitian, Jaro Winkler memberikan hasil yang paling baik dibandingkan beberapa algoritma yang telah disebutkan [11]. Beberapa peneliti mengimplementasikan Algoritma Jaro Winkler dalam pembuatan sistem mereka, diantaranya adalah penggunaan Jaro Winkler untuk pengecekan ejaan dalam teks bahasa Arab [12] dan kombinasi Algoritma Jaro Winkler dan Double Metaphone untuk pencarian ayat Alquran berbasis kemiripan fonetis [5].

Berdasarkan penelitian [11], dasar dari algoritma ini memiliki tiga bagian :

1) Menghitung panjang *string*
2) Menemukan jumlah karakter yang sama dalam dua *string*
3) Menemukan jumlah transposisi

Jaro Winkler akan menghasilkan nilai skala 0 sampai 1. Nilai 0 menandakan tidak adanya kemiripan antar *string*, sedangkan 1 menandakan dua *string* tersebut merupakan *string* yang sama.

Pada Algoritma Jaro digunakan Persamaan 1 untuk menghitung jarak (*dj*) antara dua *string* yaitu s1 dan s2, dimana *c* adalah jumlah karakter yang sama persis; |s1| adalah panjang *string* 1; |s2| adalah panjang *string* 2; *t*: jumlah transposisi; *m*: jumlah karakter yang cocok antar kedua *string*. Jarak teoritis dua buah karakter yang dianggap sama dapat dikatakan benar jika tidak melebihi batas seperti yang tercantum pada Persamaan 2.

$$d_j = \frac{1}{3}\left(\frac{m}{s_1} + \frac{m}{s_2} + \frac{m-t}{m}\right) \tag{1}$$

Jarak teoritis dua buah karakter yang disamakan dapat dibenarkan jika tidak melebihi :

$$JarakTeritoris = \left[\frac{max(|s_1|, |s_2|)}{2}\right] - 1 \tag{2}$$

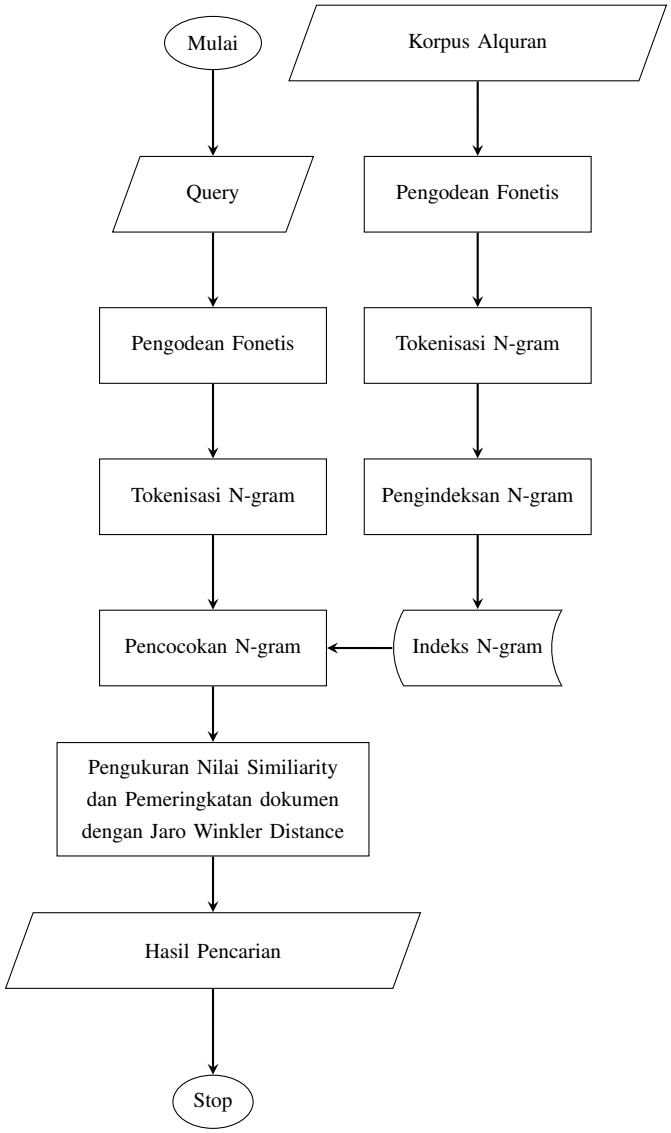
Perhitungan Jaro Winkler (*dw*) dilakukan setelah nilai *dj* ditemukan. Pada perhitugan Jaro Winkler terdapat variabel baru yaitu prefix length (l) yang menunjukkan panjang awalan yaitu panjang karakter yang sama terhadap *string* yang dibandingkan sampai ditemukan ketidaksamaan. Prefix scale (*p*) merupakan konstanta bobot. Winkler menggunakan nilai p = 0.1 dan l kurang dari atau sama dengan 4.

$$d_w = d_j + (lp(1 - d_j)) \tag{3}$$

III. SISTEM PENCARIAN FONETIK ALQURAN

A. Gambaran Umum Sistem

Dalam pembuatan aplikasi pencarian potongan ayat Alquran berdasarkan pengucapan ini dilakukan beberapa proses yang berkaitan dengan penambangan teks. Diantaranya yaitu ada proses pengodean fonetis, tokenisasi N-gram, pengindeksan N-gram, pencocokan N-gram, pengukuran nilai *similiarity* dan pemeringkatan dokumen. Setiap proses yang dilakukan melibatkan dataset yang telah ditentukan. Adapun *flowchart*-nya terdapat pada **Gambar 1**.



Gambar 1: Flowchart Sistem

B. Dataset

Dataset yang digunakan pada penelitian ini adalah korpus teks Alquran dengan aksara Latin atau Alquran transliterasi Latin sesuai dengan pelafalan orang indonesia. Pada penelitian ini, *dataset* yang digunakan adalah seluruh surah dan juz dalam Alquran. *Dataset* tersebut diperoleh dari Tanzil. Data berupa *text file* yang setiap barisnya berisi satu ayat dalam bahasa dan aksara arab tanpa tanda baca.

C. Pengodean Fonetis

Algoritma pengodean fonetis yang digunakan pada penelitian ini yaitu algoritma yang dibuat di Lafzi [1], yaitu dengan cara mempertimbangkan cara membaca Alquran (tajwid) yang sedikit berbeda dengan cara membaca teks berbahasa Arab biasa. Selain itu, untuk mengantisipasi kesalahan cara pelafalan Al-quran yang sering terjadi, dipertimbangkan juga kesalahan yang umum terjadi tersebut. Contoh kesalahan tersebut ialah pelafalan huruf ش dengan lafal س. Pengodean fonetis diterapkan baik pada teks Al-Quran maupun pada *query* yang dimasukkan oleh pengguna. Untuk teks Alquran, dilakukan pengodean fonetis dengan prosedur secara berurutan sebagai berikut:

1) Penghilangan spasi
   Seluruh karakter spasi dalam Alquran dihilangkan.
2) Penghilangan *syaddah*
   Bunyi huruf konsonan ganda yang ditandai dengan tanda *syaddah* dihilangkan.
3) Penggabungan konsonan mati
   Sama dengan penghilangan syaddah, dua huruf konsonan yang sama dan berdampingan, jika huruf konsonan pertama mati (*sukun*), maka huruf konsonan yang mati tersebut dihilangkan.
4) Penanganan akhir ayat
   Semua huruf di akhir ayat memiliki cara baca yang khusus dengan aturan tertentu.
5) Subtitusi *tanwin*
   Tanda vokal *tanwin* diganti dengan huruf *nun* mati untuk penyeragaman.
6) Penghilangan bacaan panjang
   Bacaan panjang dipendekkan karena secara umum dapat menimbulkan kesalahan pelafalan.
7) Penghilangan huruf yang tidak dibaca
   Seluruh huruf yang tidak memiliki tanda vokal tidak dilafalkan sehingga dihilangkan.
8) Subtitusi bacaan *iqlab*
   Seluruh bacaan *iqlab* diganti dari *nun* mati menjadi *mim* mati.
9) Subtitusi bacaan *idgham*
   Seluruh bacaan *idgam* dihilangkan *nun* matinya, kecuali pada beberapa kata tertentu.
10) Pemadanan ke kode fonetis
    Setelah semua prosedur tadi selesai, seluruh karakter pada teks Alquran dipadankan dengan kode fonetis.

Prosedur pengodean tersebut belum mencakup seluruh aturan tajwid, terutama aturan tingkat lanjut. Beberapa aturan tajwid yang belum dapat ditangani antara lain *idgham mutajanisain* dan *idgham mutaqaribain* yang membutuhkan analisis fonetik yang lebih mendalam. Selain itu, ada juga aturan *nun washal* yang membutuhkan analisis linguistik serta aturan bacaan *imalah*, *isymam*, dan *saktah* yang hanya terdapat pada beberapa tempat saja di dalam Al-Quran.

Sedangkan untuk masukan *query* teks Latin dilakukan pengodean fonetis dengan prosedur secara berurutan sebagai berikut:

1) Subtitusi vokal
   Hanya terdapat tiga jenis aksara vokal dalam bahasa Arab, yaitu *A*, *I* dan *U*. Sedangkan dalam aksara Latin terdapat tambahan *E* dan *O*. Oleh karena itu, vokal disubtitusi dengan dengan huruf vokal *O* diganti dengan huruf vokal *A* dan huruf vokal *A* diganti dengan huruf vokal *I*.
2) Penggabungan konsonan
   Setara dengan langkah penghilangan huruf *syaddah*, huruf-huruf konsonan yang sama digabungkan dan dijadikan satu.
3) Penggabungan vokal
   Setara dengan langkah penghilangan bacaan panjang, huruf-huruf vokal yang dan berdampingan digabungkan dan dijadikan satu.
4) Subtitusi diftong
   Subtitusi diftong dilakukand dengan mengubah *AI* menjadi *AY* dan *AU* menjadi *AW*.
5) Penandaan huruf *hamzah*
   Seluruh huruf *hamzah* ditandai.
6) Subtitusi bacaan *ikhfa*

Bacaan *ikhfa* biasanya dituliskan. dengan bunyi *NG* sehingga huruf *G* harus dihilangkan agar setara dengan pengodean teks Alquran.

7) Subtitusi bacaan *iqlab*
   Bacaan *iqlab* diproses dengan mengubah susunan huruf konsonan *NB* menjadi *MB*.
8) Subtitusi bacaan *idgam*
   Bacaan *idgham* diproses dengan menghilangkan huruf konsonan *N* jika bertemu dengan huruf-huruf *idhgam*.
9) Pemadanan ke kode fonetis
   Pemadanan ke kode fonetis dari teks Latin mempertimbangkan huruf Arab yang direpresentasikan dalam dua huruf konsonan.
10) Penghilangan spasi
   Seluruh spasi dihilangkan karena tidak memengaruhi pelafalan.

Prosedur pengodean tersebut masih memiliki kekurangan pada penandaan huruf *hamzah*. Masih terdapat ambiguitas untuk menentukan 2 huruf vokal berdampingan termasuk diftong, bacaan panjang, atau merupakan huruf hamzah.

D. Tokenisasi N-gram

Pada penelitian ini , diterapkan Algoritma N-gram. Pada *string* kode fonetis yang dihasilkan, baik dari *query* maupun teks Alquran, dilakukan tokenisasi untuk mengambil N-gram (bigram, trigram dan quadgram). N-gram yang diambil tidak memerlukan penanda awal atau akhir *string* karena *query* dapat berupa *substring* dari teks Alquran. Proses tokenisasi menggunakan *overlapping window* sepanjang 3 karakter.Sebagai contoh, trigram dari *string* "ARABIC" adalah "ARA", "RAB", "ABI", dan "BIC".

Proses tokenisasi dilakukan pada teks dari dokumen sebelum diindeks. Proses ini adalah proses pemotongan dokumen menjadi token yang kemudian dimasukkan sebagai term pada indeks [13]. Pada proses tokenisasi N-gram, tidak diperlukan segmentasi kalimat pada teks menjadi sejumlah kata, tetapi hanya perlu memotong teks dengan *overlapping window* sepanjang n.

E. Pengindeksan N-gram

Pada kode fonetis teks Alquran yang telah ditokenisasi, dilakukan pembentukan *inverted index*. *Inverted index* menggunakan N-gram sebagai term dan ayat Alquran sebagai dokumen. Satu dokumen pada indeks adalah satu ayat pada Alquran. Informasi yang disimpan pada indeks adalah identifier dokumen, jumlah N-gram tertentu pada dokumen, serta posisi kemunculan pertama N-gram pada dokumen.

Proses pengindeksan N-gram dilakukan dengan menambahkan lima trigram ayat berikutnya (*next verse*) ke ayat saat ini (*current verse*). Hasil dari pengindeksan ini digunakan untuk menangani *query* yang melintasi ayat. Pada pengindeksan, korpus kode fonetis dapat langsung ditokenisasi.

Misalkan يَوْمِ الدِيْنِ (٤) إِيَّاكَ jika ditransliterasi ke aksara Latin menjadi YAWMIDINIYAKA terdapat 11 trigram, yaitu YAW, AWM, WMI, MID, IDI, DIN, INI, NIY, IYA, YAK, dan AKA. Maka semua trigram akan dipetakan ke dalam identifier dokumen "surah index":[*trigram position*] seperti pada **Gambar I**.

Tabel I: Documents Identifier Mapping

Id	Trigram	Document Identifier
1959	YAW	"4":[7]
206	AWM	"4":[8]
1802	WMI	"4":[9]
1113	MID	"4":[10]
700	IDI	"4":[11]
387	DIN	"4":[12]
809	INI	"4":[13]
1218	NIY	"4":[14]
883	IYA	"4":[15]
1952	YAK	"4":[16]
81	AKA	"4":[17]

F. Pencocokan N-gram

N-gram dari *query* dibandingkan dengan N-gram yang ada pada indeks. Pada tahap ini, dihitung jumlah N-gram dari dokumen yang sama dengan N-gram dari *query*. Perhitungan dilakukan dengan memanfaatkan informasi yang tersimpan dalam indeks.

G. Pengukuran Nilai Similiarity dan Pemeringkatan Dokumen

Pengukuran nilai kesamaan dilakukan dengan menggunakan Algoritma Jaro Winkler. Apabila kedua kode fonetis memiliki nilai kesamaan yang tinggi atau mendekati 1, maka kedua kode tersebut dianggap cocok. Rentang nilai kesamaan adalah 0 sampai 1. Dimana 0 menandakan tidak adanya kesamaan antar srting dan 1 menandakan kedua *string* sama. Ayat dengan kode fonetis yang dianggap cocok, akan dikeluarkan sebagai hasil pencarian.

Tahapan dilakukannya pemeringkatan dari hasil pencocokan N-gram dan pengukuran nilai similiarity sebelumnya. Pengurutan dilakukan berdasarkan besar nilai similiarity dari nilai yang paling besar hingga nilai yang paling kecil.

Untuk memeringkat dokumen, perlu dilakukan pemberian skor terhadap seluruh dokumen yang didapat pada proses pencarian. Pada pengukuran kesamaan dengan N-gram, cara yang paling sederhana ialah dengan memberi skor sebanyak jumlah N-gram yang sama antara *query* dan masing-masing dokumen. Cara inilah yang digunakan pada jenis pemeringkatan pertama, yaitu pemeringkatan jumlah. Skor masing-masing dokumen adalah jumlah N-gram yang sama antara *query* dengan dokumen. Dengan demikian, skor maksimal ialah sebanyak jumlah N-gram pada *query*, sedangkan skor minimal ialah 1 (karena proses pencarian hanya mengambil dokumen yang memiliki paling sedikit 1 N-gram yang sama dengan *query*).

H. Hasil Pencarian

Keluaran dari sistem yang berupa ayat-ayat Alquran dimana berisikan informasi mengenai nama surah, nomor surah, nomor ayat, teks ayat Alquran aksara Arab, teks ayat Alquran aksara Latin dan hasil Algoritma Jaro Winkler Distance yang memiliki kemiripan kode fonetis antara *string query* masukan dengan *string* keluaran dari sistem. Jika *query* yang dimasukkan melintasi ayat, maka akan ditampilkan hasil pencarian lebih dari satu ayat sekaligus dalam satu indeks.

IV. Hasil Pengujian dan Analisis

Pengujian dan evaluasi terhadap sistem dilakukan dengan menghitung nilai MAP dan *Recall* dari keluaran hasil pencarian dengan *query* yang dimasukkan. *Query* yang dimasukkan akan beragam, mulai dari sembarang kata sampai melintasi ayat. *Dataset* yang digunakan adalah Alquran transliterasi Latin juz 1-30.

Mean Average Precision (MAP) adalah skor yang didapat dari pengukuran kinerja sistem pada serangkaian *query*. Kategori MAP yang baik terlihat dari hasil AP (*Average Precission*). Jika keseluruhan nilai AP rendah, artinya presisi sistem buruk, karena informasi relevan yang dikeluarkan tidak pada urutan yang berdekatan dan memiliki jumlah *output false positif* yang banyak. Cara mendapatkan skor MAP adalah dengan menghitung nilai rata-rata dari *average precision* pada tiap kueri [14].

$$MAP = \frac{1}{\mid Q \mid} \cdot \sum_{i=1}^{Q} AP(Q_i) \tag{4}$$

$$AP = \sum_{i=1}^{N} \left(\frac{TP_{-i}}{TPrank_{-i}} \right) \tag{5}$$

Recall merupakan probalitas informasi yang relevan yang di outputkan dibandingkan dengan jumlah informasi ke- seluruhan yang relevan. Skor dari Recall menentukan tingkat keberhasilan sistem dalam melakukan pencarian ayat. Nilai maksimum Recall adalah 1 dan minimum 0. Jika nilai Recall sistem adalah 1 artinya sistem berhasil melakukan pencarian informasi berdasarkan *query* yang ada. Berikut adalah notasi penghitungan Recall:

$$Recall = \frac{\mid (InformasiRelevan) \cap (KeluaranSistem) \mid}{\mid InfromasiRelevan \mid} \tag{6}$$

A. Data Uji (Test Set)

Sebelum menghitung nilai MAP dan Recall, terlebih dahulu ditentukan *test set* sebanyak masing-masing tiga puluh buah untuk pencarian lintas ayat dan pencarian tidak lintas ayat. Teks arab yaitu *(query)* yang dijadikan kata kunci pencarian dan aksara Latin adalah transliterasi bahasa arab menjadi Latin sesuai dengan bunyi. *Test set* merupakan surah Al-Fatihah ayat 1-7 dan Al-Baqarah ayat 1-26 yang diambil dari *www.tanzil.net*. *Test set* dijalankan pada Lafzi dan Lafzi+ (sistem ini). Kemudian hasil MAP dan Recall keduanya dibandingkan.

Tabel II: Contoh *test set* pencarian kata yang melintasi ayat

No	Teks Arab	Aksara Latin
1	نِ الرَّحِيمِ (١) الْحَمْدُ	nirahim'alhamd
2	بِ الْعَالَينَ(٢) الرَّحْمَـ	bil'alamin'arahma
3	نِ الرَّحِيمِ (٣) مَالِكِ	nirahimmalikiy
4	يَوْمِ الدِيْنِ (٤)إِيَّاكَ نَ	yawmidin'iyakan
5	سْتَعِينُ (٥) اهْدِنَا	nasta'in'ihdina

Tabel II adalah contoh *test set* pencarian kata yang melintasi ayat.

Tabel III: Contoh *test set* pencarian kata yang tidak melintasi ayat

No	Teks Arab	Aksara Latin
1	مِ اللَّـهِ الرَّحْمَـن	millahirrahman
2	لِلَّـهِ رَبِ الْعَالَمِ	lillahirabbilalam
3	مَـن الرَّحِيمِ	manirrahim
4	مَالِكِ يَوْمِ الدِيْنِ	malikiyawmidin
5	نَعْبُدُ وَإِيَّاكَ	na'buduwaiyyaka

Tabel III adalah contoh *test set* pencarian kata yang tidak melintasi ayat.

B. Analisis Hasil Pengujian

Tabel IV: Results of system testing for cross-verses search

Sistem	Lintas Ayat		Tidak Lintas Ayat	
	MAP	Recall	MAP	Recall
Lafzi	N/A	N/A	0.9	0.97
Lafzi+ (sistem ini)	**0.9**	**0.93**	0.9	0.97

Berdasarkan hasil pengujian pada **Tabel IV** ditemukan bahwa pencarian ayat Alquran berdasarkan *query* lintas ayat memberikan hasil yang sangat baik. Hasil tersebut berupa nilai MAP 0,9 dan nilai Recall 0,93. Hasil tersebut merupakan kemajuan atau pengembangan dari Lafzi yang mendapatkan nilai

MAP *N/A* dan nilai Recall *N/A*. Hasil pengujian lafzi mendapatkan nilai MAP *N/A* dan nilai Recall *N/A* karena dipengaruhi oleh tidak adanya fitur pencarian lintas ayat pada sistem tersebut. Sedangkan untuk pencarian dengan *query* yang tidak melintasi ayat, nilai MAP dan Recall sistem ini sama persis dengan lafzi, yaitu 0,9 dan 0,97. Hal tersebut membuktikan tidak adannya penurunan performa pencarian.

Hasil pengujian menunjukkan nilai MAP dan Recall tidak sempurna karena *query* lintas ayat yang terlalu panjang cenderung memberikan hasil yang tidak begitu akurat. Hal tersebut terjadi karena teknik yang digunakan yaitu penambahan trigram ayat berikutnya (*next verse*) ke ayat saat ini (*current verse*) terbatas hanya lima trigram. Jadi setelah melewati lima trigram, semakin panjang potongan *query* ayat berikutnya, maka semakin kecil pula nilai AP dan Recall yang dihasilkan.

V. Kesimpulan

Berdasarkan analisis hasil pengujian, dapat disimpulkan bahwa pembangunan sistem pencarian ayat Alquran lintas ayat berdasarkan kesamaan fonetis yang disesuaikan dengan aksara Latin masyarakat muslim Indonesia dapat dilakukan dengan sangat baik menggunakan Algoritma N-gram dan Jaro Winkler Distance. Hal tersebut dibuktikan pada hasil pengujian yang memperoleh nilai MAP 0,9 dan nilai Recall 0,93. Sedangkan pencarian sembarang kata mendapatkan skor MAP dan Recall sama dengan Lafzi, yang artinya tidak terjadi penurunan performa. Ditambah lagi sistem ini merupakan sistem pencarian ayat Alquran beraksara Latin pertama yang dibuat. Di samping itu, akurasi pencarian untuk *query* yang tidak melintasi ayat tidak mengalami penurunan. Untuk pencarian lintas ayat yang potongan *query* ayat berikutnya lebih dari lima trigram sebaiknya dilakukan penelitian lanjutan agar nilai AP dan Recall yang dihasilkan dapat tetap baik. Salah satu teknik yang direkomendasikan yaitu dengan menandai hasil pencarian lintas ayat dan menjadikan hasil pencarian sebagai kandidat hasil, kemudian melakukan pencarian lagi terhadap kandidat hasil tersebut.

Ucapan Terima Kasih

Penulis ingin mengucapkan terima kasih kepada para pengulas atas tanggapan mereka.

Pustaka

[1] Istiadi MA. Sistem Pencarian Ayat Al-Quran Berbasis Kemiripan Fonetis. Final Project IPB. Bogor; 2012.
[2] Wardani ZU. Analisis Aplikasi Pencarian Berdasarkan Ucapan Ayat Al-QurâĂŹan Berbasis Web Menggunakan Algoritma Soundex dan Metode Cosine Similarity. Universitas Telkom. Bandung; 2018.
[3] Arsaningtyas PA. Sistem Pencarian Ayat Al-Quran Berdasarkan Kemiripan Ucapan Menggunakan Algoritma Soundex dan Damerau-Levenshtein Distance. Universitas Telkom. Bandung; 2018.
[4] Putri E. Pembangunan Sistem Pencarian Ayat Al-QurâĂŹan Berbasis Mobile Menggunakan Algoritma Metaphone Berdasarkan Kemiripan Pengucapan dan Algoritma Levenshtein Distance untuk Mengukur Tingkat Similarity. Universitas Telkom. Bandung; 2018.
[5] Hastuti A. Sistem Pencarian Ayat Al-QurâĂŹan Berdasarkan Kemiripan Fonetis Menggunakan Algoritma Double Metaphone dan Jaro Winkler. Universitas Telkom. Bandung; 2018.
[6] Syaroni M, Munir R. Pencocokan String Berdasarkan Kemiripan Ucapan (Phonetic String Matching) dalam Bahasa Inggris. In: Seminar Nasional Aplikasi Teknologi Informasi (SNATI); 2005. .
[7] Cavnar WB, Trenkle JM, et al. N-gram-based text categorization. In: Proceedings of SDAIR-94, 3rd annual symposium on document analysis and information retrieval. vol. 161175; 1994. .
[8] Hamzah A. Deteksi Bahasa untuk dokumen teks berbahasa indonesia. In: Seminar Nasional Informatika (SEMNASIF). vol. 1; 2015. .
[9] Hsu W, Du M. Computing a longest common subsequence for a set of strings. BIT Numerical Mathematics. 1984;24(1):45–59.
[10] Cohen W, Ravikumar P, Fienberg S. A comparison of string metrics for matching names and records. In: Kdd workshop on data cleaning and object consolidation. vol. 3; 2003. p. 73–78.
[11] Rochmawati Y, Kusumaningrum R. Studi Perbandingan Algoritma Pencarian String dalam Metode Approximate String Matching untuk Identifikasi Kesalahan Pengetikan Teks. Jurnal Buana Informatika. 2016;7(2).
[12] Gueddah H, Yousfi A, Belkasmi M. The filtered combination of the weighted edit distance and the Jaro-Winkler distance to improve spellchecking Arabic texts. In: Computer Systems and Applications (AICCSA), 2015 IEEE/ACS 12th International Conference of. IEEE; 2015. p. 1–6.

[13] Manning C, Raghavan P, Schütze H. Introduction to information retrieval. Natural Language Engineering. 2008;16(1):100–103.
[14] Han S, He D, Yue Z, Jiang J. Contextual support for collaborative information retrieval. In: Proceedings of the 2016 ACM on Conference on Human Information Interaction and Retrieval. ACM; 2016. p. 33–42.