Studi Perbandingan Algoritma Pencarian *String* dalam Metode *Approximate String Matching* untuk Identifikasi Kesalahan Pengetikan Teks

Yeny Rochmawati[1], Retno Kusumaningrum[2]
Jurusan Ilmu Komputer/Informatika, Fakultas Sains dan Matematika, Universitas Diponegoro
Jl. Prof. H. Soedarto, SH. Tembalang, Semarang 50275, Jawa Tengah
E-mail: [1]yeny.rochmawati@if.undip.ac.id, [2]retno_ilkom@undip.ac.id

Masuk: 30 September 2015; Direvisi: 16 Oktober 2015; Diterima: 19 Oktober 2015;

Abstract. *Error typing resulting in the change of standard words into non-standard words are often caused by misspelling. This can be addressed by developing a system to identify errors in typing. Approximate string matching is one method that is widely implemented to identify error typing by using several string search algorithms, i.e. Levenshtein Distance, Hamming Distance, Damerau Levenshtein Distance and Jaro Winkler Distance. However, there is no study that compares the performance of the four algorithms. Therefore, this research aims to compare the performance between the four algorithms in order to identify which algorithm is the most accurate and precise in the search string based on various errors typing. Evaluation is performed by using users' relevance judgments which produce the mean average precision (MAP) to determine the best algorithm. The result shows that Jaro Winkler Distance algorithm is the best in word-checking with 0.87 of MAP value when identifying the typing error of 50 incorrect words.*
Keywords: *Errors typing, Levenshtein, Hamming, Damerau Levenshtein, Jaro Winkler*

Abstrak. *Kesalahan pengetikan mengakibatkan kata baku berubah menjadi kata tidak baku karena ejaan yang digunakan tidak sesuai. Hal tersebut dapat ditangani dengan mengembangkan sistem untuk mengidentifikasi kesalahan pengetikan. Metode approximate string matching merupakan salah satu metode yang banyak diterapkan untuk mengidentifikasi kesalahan pengetikan dengan berbagai jenis algoritma pencarian string yaitu Levenshtein Distance, Hamming Distance, Damerau Levenshtein Distance dan Jaro Winkler Distance. Akan tetapi studi perbandingan kinerja dari keempat algoritma tersebut untuk Bahasa Indonesia belum pernah dilakukan. Oleh karena itu penelitian ini bertujuan untuk melakukan studi perbandingan kinerja dari keempat algoritma tersebut sehingga dapat diketahui algoritma mana yang lebih akurat dan tepat dalam pencarian string berdasarkan kesalahan penulisan yang bervariasi. Evaluasi yang dilakukan menggunakan user relevance judgement yang menghasilkan nilai mean average precision (MAP) untuk menentukan algoritma yang terbaik. Hasil penelitian terhadap 50 kata salah menunjukkan bahwa algoritma Jaro Winkler Distance terbaik dalam melakukan pengecekan kata dengan nilai MAP sebesar 0,87.*
Kata Kunci: *Kesalahan pengetikan, Levenshtein, Hamming, Damerau Levenshtein, Jaro Winkler*

1. Pendahuluan

Keterampilan menulis merupakan keterampilan yang penting dalam kehidupan, baik dalam kehidupan pendidikan maupun masyarakat. Menulis adalah kegiatan menyampaikan gagasan yang tidak dapat secara langsung diterima dan direaksi oleh pihak yang dituju. Aktivitas menulis merupakan salah satu manisfestasi kemampuan dan keterampilan berbahasa paling akhir yang dikuasai pengguna bahasa setelah mendengarkan, membaca, dan berbicara (Luqman, 2009).

Kesalahan pengetikan yang sering terjadi dapat menyebabkan kata baku berubah menjadi kata tidak baku karena ejaan yang digunakan tidak sesuai. Beberapa faktor yang menyebabkan terjadinya kesalahan pengetikan adalah letak huruf pada *keyboard* yang berdekatan, kesalahan karena *slip* pada tangan atau jari, kesalahan yang disebabkan oleh ketidaksengajaan. Proses pengecekan kesalahan pengetikan dengan cara manual akan menghabiskan banyak waktu. Oleh karena itu diperlukan suatu sistem yang mampu mengidentifikasi kesalahan pengetikan agar sesuai dengan kaidah penulisan EYD yang benar. Sehingga kesalahan-kesalahan penulisan dapat dihindari. Salah satu pendekatan yang dapat dilakukan untuk mengidentifikasi kesalahan pengetikan adalah metode pencocokan *string*.

Pada aplikasi Microsoft Word terdapat editor teks untuk pencocokan *string* hanya saja penggunaannya belum maksimal karena *user* harus memilih kata yang salah kemudian mengecek kata menggunakan *spelling & grammar* untuk mendapatkan kata yang benar. Sedangkan *user* cenderung tidak merasa apabila terdapat kesalahan pengetikan walaupun telah ditandai dengan garis merah di bawah kata yang salah sehingga kembali lagi *user* harus mengecek semua teks secara manual. Sistem untuk identifikasi kesalahan pengetikan ini merupakan sistem berbasis web yang dapat digunakan oleh masyarakat luas dengan didukung adanya kamus besar bahasa Indonesia sehingga jika terdapat kata yang tidak sesuai dengan kamus akan dapat terdeteksi sebagai kata salah dan sistem akan mengeluarkan semua kata salah yang ditemukan beserta kata saran untuk perbaikannya.

Pencocokan *string* secara garis besar dibedakan menjadi dua yaitu *exact string matching* (pencocokan *string* secara tepat dengan susunan karakter dalam *string* yang dicocokkan) dan *inexact string matching* (pencocokan *string* secara samar, yaitu pencocokan *string* dimana *string* yang dicocokkan memiliki kemiripan namun keduanya memiliki susunan karakter yang berbeda) (Sagita & Prasetiyowati, 2012). Dua pendekatan *inexact string matching* meliputi *approximate string matching* yang mencocokkan *string* berdasarkan kemiripan penulisan dan *phonetic string matching* yang mencocokkan *string* berdasarkan kemiripan ucapan. *Approximate string matching* dapat digunakan untuk pencarian *string* berdasarkan *string* yang sama dan *string* yang memiliki kemiripan penulisan dengan *string* yang terdapat pada kamus. Metode ini dapat digunakan untuk pencarian kata tidak baku karena dapat mengidentifikasi *string* yang sama dan yang memiliki kemiripan penulisan. Beberapa algoritma yang dapat digunakan untuk pencarian *string* dalam *approximate string matching* diantaranya adalah *Hamming Distance, Levenshtein Distance, Damerau Levenshtein Distance* dan *Jaro Winkler Distance*.

Berdasarkan latar belakang tersebut, permasalahan yang muncul adalah bagaimana perbandingan kinerja dari algoritma *Hamming Distance, Levenshtein Distance, Damerau Levenshtein Distance* dan *Jaro Winkler Distance* dalam algoritma *string matching* untuk mengidentifikasi kesalahan pengetikan teks Bahasa Indonesia? Adapun perbandingan kinerja keempat algoritma menggunakan nilai *Mean Average Precision* (MAP) sebagai indikator.

2. Tinjauan Pustaka

2.1. Kata Baku dalam Bahasa Indonesia

Bahasa baku merupakan salah satu bentuk ragam bahasa yang tercermin dari penggunaan kaidah yang benar meliputi ejaan, lafal, struktur dan pemakaiannya. Bahasa baku digunakan dalam situasi resmi yang menjadi acuan atau model oleh masyarakat pemakai bahasa. Bahasa Indonesia baku termasuk salah satu ragam bahasa Indonesia yang bentuk bahasanya telah diterima, difungsikan dan dipakai sebagai model oleh masyarakat Indonesia secara luas. Oleh karena itu kata baku dalam bahasa baku sangat penting penggunaannya.

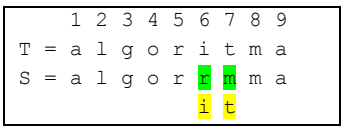
2.2. Klasifikasi Pencocokan *String*

Pencocokan *string* (*string matching*) secara garis besar dapat dibedakan menjadi dua (Sagita & Prasetiyowati, 2012) yaitu: (1) *Exact string matching*, merupakan pencocokan *string* secara tepat dengan susunan karakter dalam *string* yang dicocokkan memiliki jumlah maupun urutan karakter dalam *string* yang sama. Misalnya, kata obat akan menunjukkan kecocokan

hanya dengan kata obat. (2) *Inexact string matching* atau *fuzzy string matching*, merupakan pencocokan *string* secara samar yaitu pencocokan *string* dimana string yang dicocokkan memiliki kemiripan namun keduanya memiliki susunan karakter yang berbeda (mungkin jumlah atau urutannya) tetapi *string* tersebut memiliki kemiripan baik kemiripan tekstual/penulisan atau kemiripan ucapan. Pencocokan *string* berdasarkan kemiripan tekstual/penulisan meliputi jumlah karakter, susunan karakter dalam dokumen disebut sebagai *approximate string matching* sedangkan pencocokan *string* berdasarkan kemiripan ucapan dari segi pengucapan disebut sebagai *phonetic string matching*. Berikut ini contoh dari *approximate string matching* dan *phonetic string matching*: (a) panitea dengan panitia, memiliki jumlah karakter yang sama tetapi ada karakter yang berbeda. Jika perbedaan karakter ini dapat ditoleransi sebagai sebuah kesalahan penulisan maka dua *string* tersebut dikatakan cocok. (b) obat dengan obad dari tulisan berbeda tetapi dalam pengucapannya mirip sehingga dua *string* tersebut dianggap cocok. Contoh yang lain adalah obat, dengan obbat, obaat, obatt, obate.

2.3. Algoritma *Hamming Distance*

Cara kerja algoritma *Hamming Distance* yaitu dengan mengukur jarak antara dua *string* yang ukurannya sama dengan membandingkan karakter yang terdapat pada kedua *string* pada posisi yang sama (Hamming, 1950). *Hamming Distance* dari dua *string* adalah jumlah simbol dari kedua *string* yang berbeda. Operasi yang digunakan hanya operasi penggantian huruf. Contoh ilustrasi algoritma *Hamming Distance* ditunjukkan pada Gambar 1.



Gambar 1. Ilustrasi Algoritma *Hamming Distance*

2.4. Algoritma *Levenshtein Distance*

Algoritma *Levenshtein Distance* ditemukan oleh Vladimir Levenshtein, seorang ilmuan asal Rusia pada tahun 1965, algoritma ini sering juga disebut dengan *Edit Distance*. Algoritma *Levenshtein Distance* bekerja dengan menghitung jumlah minimum pentranformasian suatu *string* menjadi *string* lain yang meliputi penghapusan, penyisipan, dan penggantian (Mulyanto, 2010). Langkah-langkah untuk algoritma *Levenshtein Distance* (Setiadi, 2013) seperti pada Kode 1. Diberikan contoh kata yang salah yaitu tekhnologi dengan kata yang benar yaitu teknologi. S: tekhnologi → m=10; T: teknologi → n=9.

Kode 1. Algoritma *Levenshtein Distance*

```
for i ← 1 to m do { source prefixes initialization }
   d[i][0] ← i
endfor
for j ← 1 to n do { target prefixes initialization }
   d[0][j] ← j
endfor
{ using Levenshtein Algorithm to check }
for i ← 1 to n do
   for j ← 1 to m do
        if (s[i] == t[j]) then
                d[i][j] ← d[i-1][j-1] {same character}
        else
             d[i][j] ← minimum
               (
                 d[i-1][j] + 1, { deletion }
                 d[i][j-1] + 1, { insertion }
                 d[i-1][j-1] + 1 { substitution }
               )
        endif
   endfor
end for
→ d[m][n] { return results }
```

Untuk menghitung jarak pada *Levenshtein Distance* dapat dilakukan menggunakan tabel seperti pada Tabel 1. Pada tabel 1 terdapat sel sejumlah huruf yang akan dicek. Pengisian tabel dimulai dari sel (1,1) dengan posisi baris sebagai indeks *i* dan posisi kolom sebagai indeks *j*. Pengisian tabel berdasarkan algoritma *Levenshtein Distance* pada Kode 1.

Tabel 1. Perhitungan *Levenshtein Distance*

			String Target (*T*)								
			T	**E**	**K**	**N**	**O**	**L**	**O**	**G**	**I**
		0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
String Sumber (*S*)	**T**	**1**	0	1	2	3	4	5	6	7	8
	E	**2**	1	0	1	2	3	4	5	6	7
	K	**3**	2	1	0	1	2	3	4	5	6
	H	**4**	3	2	1	1	2	3	4	5	6
	N	**5**	4	3	2	1	2	3	4	5	6
	O	**6**	5	4	3	2	1	2	3	4	5
	L	**7**	6	5	4	3	2	1	2	3	4
	O	**8**	7	6	5	4	3	2	1	2	3
	G	**9**	8	7	6	5	4	3	2	1	2
	I	**10**	9	8	7	6	5	4	3	2	1

2.5. Algoritma *Damerau Levenshtein Distance*

Damerau Levenshtein adalah sebuah fungsi pada *finite string* dari sebuah alphabet ke *integer*. Sebuah matriks jarak yang diberikan *string* S_1, S_2, S_3 yang memenuhi kondisi (Sutisna & Adisantoso, 2010): (1) *Non-negativity*: $d(S_1,S_2) \geq 0$ (2) *Non-degeneracy*: $d(S_1,S_2) = 0$ jika dan hanya jika $S_1 = S_2$ (3) *Symmetry*: $d(S_1,S_2) = d(S_2,S_1)$ (4) *Triangle Inequality*: $d(S_1,S_2) + d(S_2,S_3) \geq d(S_1,S_3)$.

Algoritma *Damerau Levenshtein Distance* melakukan operasi perbandingan kata-kata dengan memperhatikan empat macam kesalahan pengetikan (misalnya kata DAMERAU) (Sutisna & Adisantoso, 2010) yaitu: (1) Penyisipan atau penambahan sebuah huruf, misalnya DAHMERAU. (2) Penghapusan sebuah huruf, misalnya DAMRAU. (3) Penggantian sebuah huruf dengan huruf lain, misalnya DANERAU. (4) Penukaran sebuah huruf berurutan, misalnya DAMERUA. Langkah-langkah untuk algoritma *Damerau Levenshtein Distance* (Setiadi, 2013) seperti pada Kode 2. Diberikan contoh kata yang salah yaitu yagn dengan kata yang benar yaitu yang. *S*: yagn → *m*=4; *T*: yang → *n*=4.

Kode 2. Algoritma *Damerau Levenshtein Distance*

```
for i ← 1 to m do { source prefixes initialization }
   d[i][0] ← i
endfor
for j ← 1 to n do { target prefixes initialization }
   d[0][j] ← j
endfor
{ using Damerau Levenshtein Algorithm to check }
for i ← 1 to n do
   for j ← 1 to m do
      if (s[i] == t[j]) then cost ← 0
      else cost ← 1
      endif
      d[i][j] ← minimum (
         d[i-1][j] + 1, { deletion }
         d[i][j-1] + 1, { insertion }
         d[i-1][j-1] + 1 { substitution }
       )
       if (i > 1 and j > 1 and s[i] == t[j-1] and s[j-1] == t[i]) then
          d[i][j] ← minimum (
               d[i][j],
               d[i-2][j-2] + cost { transposition }
           )
       endif
   endfor
end for
→ d[m][n] { return results }
```

Untuk menghitung jarak pada *Damerau Levenshtein Distance* dapat dilakukan menggunakan tabel seperti yang ditunjukkan pada Tabel 2. Pada tabel 2 terdapat sel sejumlah huruf yang akan dicek. Pengisian tabel dimulai dari sel (1,1) dengan posisi baris sebagai indeks *i* dan posisi kolom sebagai indeks *j*. Pengisian tabel berdasarkan algoritma *Damerau Levenshtein Distance* pada Kode 2.

Tabel 2. Perhitungan *Damerau Levenshtein Distance*

			String Target (*T*)			
			Y	**A**	**N**	**G**
		0	**1**	**2**	**3**	**4**
String Sumber (*S*)	**Y**	**1**	0	1	2	3
	A	**2**	1	0	1	2
	G	**3**	2	1	1	1
	N	**4**	3	2	1	1

2.6. Algoritma *Jaro Winkler Distance*

Algoritma *Jaro Winkler Distance* merupakan varian dari *Jaro Distance metric* yaitu sebuah algoritma untuk mengukur kesamaan antara dua *string*, biasanya algoritma ini digunakan di dalam pendeteksian duplikat. Semakin tinggi *Jaro Winkler Distance* untuk dua *string* maka semakin mirip dengan *string* tersebut. Nilai normalnya ialah nol menandakan tidak ada kesamaan dan dkk yang menandakan adanya kesamaan (Kornain, dkk., 2014). Dasar dari algoritma ini memiliki tiga bagian: (1) Menghitung panjang *string*. (2) Menemukan jumlah karakter yang sama di dalam dua *string*. (3) Menemukan jumlah transposisi. Pada algoritma *Jaro* digunakan Persamaan 1 untuk menghitung jarak (dj) antara dua *string* yaitu s_1 dan s_2 (Khatami, 2013), dimana *c*: jumlah karakter yang sama persis; $|s_1|$: panjang *string* 1; $|s_2|$: panjang *string* 2; *t*: jumlah transposisi. Jarak teoritis dua buah karakter yang dianggap sama dapat dikatakan benar jika tidak melebihi batas seperti yang tercantum pada Persamaan 2.

Jaro Winkler Distance menggunakan *prefix scale* (*p*) yang memberikan tingkat penilaian yang lebih dan *prefix length* (ℓ) yang menyatakan panjang awalan yaitu panjang karakter yang sama dengan *string* yang dibandingkan sampai ditemukan ketidaksamaan, jumlah karakter yang sama sebelum ditemukan ketidaksamaan adalah maksimal 4. Bila *string* s_1 dan s_2 yang dibandingkan, maka *Jaro Winkler Distance*-nya ditentukan dengan Persamaan 3, dimana *dj*: Jaro Distance untuk string s_1 dan s_2; ℓ*:* panjang prefiks umum di awal string (panjang karakter yang sama sebelum ditemukan ketidaksamaan max 4); *p*: konstanta *scaling factor* (p=0,1).

$$\text{dj} = \frac{1}{3}\left(\frac{c}{|s_1|} + \frac{c}{|s_2|} + \frac{c-t}{c}\right) \quad (1)$$

$$\text{jarak teoritis} = \left(\frac{\max(|s_1|, |s_2|)}{2}\right) - 1 \quad (2)$$

$$\text{dw} = dj + (\ell p(1 - dj)) \quad (3)$$

Diberikan contoh kata yang salah yaitu kosnultasi dengan kata yang benar yaitu konsultasi. Berikut perhitungan manual nilai *distance*-nya. $s_1 = kosnultasi$ $s_2 = konsultasi$ maka nilai $c = 10$, $s_1 = 10$, dan $s_2 = 10$. Selanjutnya karena karakter yang tertukar hanyalah S dan N maka *t*=1, sehingga nilai *Jaro distance*-nya adalah:

$$\text{dj} = \frac{1}{3}\left(\frac{c}{|s_1|} + \frac{c}{|s_2|} + \frac{c-t}{c}\right); \ \text{dj} = \frac{1}{3}\left(\frac{10}{10} + \frac{10}{10} + \frac{10-1}{10}\right) = 0.967$$

Bila diperhatikan susunan s_1 dan s_2 dapat diketahui nilai *prefix length* (ℓ)=2 karena karakter yang sama sebelum ditemukan kesalahan adalah karakter ‘k’ dan karakter ‘o’. Nilai konstanta *scaling factor p*=0,1. Maka nilai *Jaro Winkler Distance* adalah:

$$\text{dw} = dj + (\ell p(1 - dj)) \ ; \ \text{dw} = 0.967 + (2x0.1(1 - 0.967)) = 0.9736$$

Oleh karena itu nilai *distance* untuk *Jaro Winkler* pada kata yang salah yaitu kosnultasi dan kata kamus yang menjadi target yaitu konsultasi adalah 0,9736.

2.7. Ekstraksi Kata

Proses ekstraksi kata diperlukan untuk proses pengambilan kata yang sesuai yang akan dicocokkan dengan kamus. Tahapan proses ekstraksi kata yang digunakan pada aplikasi ini hanyalah proses *tokenization*. *Tokenization* adalah tugas memisahkan deretan kata di dalam kalimat paragraf atau halaman menjadi token atau potongan kata tunggal. Tahapan ini juga menghilangkan karakter-karakter tertentu seperti tanda baca dan mengubah semua token ke bentuk huruf kecil (*lower case*) yang disebut *case folding* (Pyriana, 2011).

2.8. *Mean Average Precision*

Mean average precision memberikan sebuah nilai tunggal terhadap seluruh titik *recall* dari seluruh pengukuran dan sudah terbukti dapat menunjukkan tingkat perbedaan dan stabilitas yang baik. *Mean average precision* akan dihitung terhadap sejumlah k kata saran yang di-*retrieve* dan relevan, dan angkanya akan dirata-rata sesuai dengan kebutuhan.

3. Metode Penelitian

3.1. Pembentukan Kamus

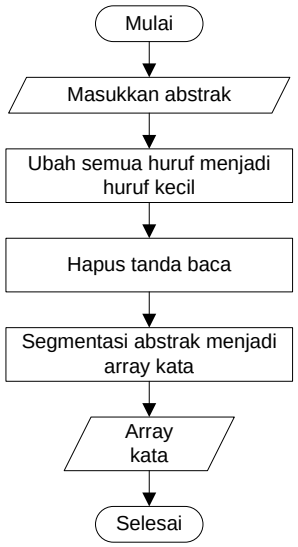
Data yang digunakan dalam penelitian ini berupa kamus besar bahasa Indonesia yang memiliki kata berjumlah kurang lebih 41.168 kata. Kata yang terdapat di dalam kamus tersebut meliputi kata dasar, kata ber-awalan, kata ber-akhiran, kata ber-imbuhan, kata sambung, kata depan dan jenis kata yang termasuk dalam kata baku bahasa Indonesia. Penelitian ini juga menggunakan kamus istilah komputer yang berisi kata-kata istilah dalam bidang komputer seperti kata *framework*, PHP, *bookmark*, *mouse* dan lain-lain.

3.2. Pengumpulan Abstrak

Data abstrak yang digunakan adalah abstrak tulisan ilmiah (laporan, makalah, karya ilmiah, skripsi) sebanyak lima abstrak dengan ketentuan jumlah kata pada abstrak tersebut tidak boleh melebihi 200 kata.

3.3. *Preprocessing*

Proses ini merupakan proses ekstraksi kata. Semua huruf dalam abstrak diubah menjadi huruf kecil (*lowercase*), hanya huruf ‘a’ sampai dengan huruf ‘z’ yang diterima. Diagram alir untuk tahap *preprocessing* tersebut dapat dilihat pada Gambar 2.



Gambar 2. Diagram Alir *Preprocessing*

3.4. Pencocokan Kata berdasarkan *Distance*

Proses ini merupakan proses pencocokan kata berdasarkan empat algoritma yaitu *Hamming Distance, Levenshtein Distance, Damerau Levenshtein Distance* dan *Jaro Winkler Distance.* Kata salah yang digunakan sebanyak 50 kata untuk dilakukan pengecekan langsung dan pengecekan berdasarkan empat jenis kesalahan kata yaitu kesalahan penghapusan huruf, penambahan huruf, penggantian huruf dan penukaran huruf. Masing-masing *distance* memiliki data masukan yang sama setelah proses pencocokan kata tiap *distance* yaitu saran kata sesuai kamus dan nilai *distance* yang dihasilkan. Data masukan tersebut diurutkan berdasarkan nilai *distance*, jika terdapat kata dengan nilai *distance* sama maka urutannya berdasarkan nilai *distance* kemudian alfabet.

Proses pengurutan yang dilakukan pada algoritma *Hamming Distance, Levenshtein Distance* dan *Damerau Levenshtein Distance* sama yaitu dengan mencari nilai terkecil dari setiap kesalahan yang ada. Proses pengurutan pada algoritma *Jaro Winkler Distance* berbeda dengan ketiganya yaitu mencari nilai *distance* yang terbesar. Pada algoritma *Jaro Winkler Distance* kata dianggap sama persis ketika nilai *distance* adalah satu sedangkan untuk ketiga algoritma yang lain kata dianggap sama persis ketika nilai *distance* adalah nol. Sehingga proses pengurutan untuk algoritma *Hamming Distance, Levenshtein Distance* dan *Damerau Levenshtein Distance* dimulai dari *distance* yang nilainya paling kecil, sedangkan pada algoritma *Jaro Winkler Distance* proses pengurutannya dimulai dari nilai *distance* yang terbesar.

3.5. Evaluasi

Evaluasi dilakukan untuk mengetahui algoritma mana yang memiliki kinerja terbaik dalam aplikasi ini menggunakan pendekatan *user relevance judgement* untuk menentukan relevan atau tidaknya kata saran yang dihasilkan masing-masing algoritma pada sistem tersebut. Evaluasi yang dilakukan menggunakan *precision recall* yang akan menghasilkan suatu nilai *average precision* dan *mean average precision*. Pada kondisi pengecekan satu kata salah yang menghasilkan 10 kata saran yang sesuai. Proses tersebut berjalan sampai kata ke-50. Nilai *precision* yang dihasilkan akan digunakan untuk menentukan nilai *average precision* untuk masing-masing kata salah, selanjutnya nilai tersebut digunakan untuk menentukan nilai MAP dengan menghitung rata-rata dari keseluruhan *average precision*.

4. Hasil Eksperimen dan Analisa

4.1. Data

Data yang dimaksud adalah data yang digunakan dalam eksperimen. Sehingga data yang digunakan berupa koleksi kata yang salah berdasarkan tipe kesalahan untuk pengujian. Koleksi kata yang salah meliputi kesalahan penulisan karena penghapusan huruf yang ditunjukkan pada Tabel 3, kesalahan penulisan karena penambahan huruf yang ditunjukkan pada Tabel 4, kesalahan penulisan karena penggantian huruf yang ditunjukkan pada Tabel 5 dan kesalahan penulisan karena penukaran huruf yang ditunjukkan pada Tabel 6.

Tabel 3. Kesalahan Penulisan karena Penghapusan Huruf

No	Kata ke-	Kata Salah	Kata yang benar
1.	Kata 1	sat	satu
2.	Kata 2	sehinga	sehingga
3.	Kata 3	mempoduksi	memproduksi
4.	Kata 4	pendatan	pendataan
5.	Kata 5	dwngn	dengan
6.	Kata 6	pemograman	pemrograman
7.	Kata 7	mengunakan	menggunakan
8.	Kata 8	penorganisasian	pengorganisasian
9.	Kata 9	mrangkul	merangkul
10.	Kata 10	pelangan	pelanggan

Tabel 4. Kesalahan Penulisan karena Penambahan Huruf

No	Kata ke-	Kata Salah	Kata yang benar
1.	Kata 1	oleha	oleh
2.	Kata 2	pesertta	peserta
3.	Kata 3	perusahaann	perusahaan
4.	Kata 4	menyimapan	menyimpan
5.	Kata 5	prograamm	program
6.	Kata 6	tekhnologi	teknologi
7.	Kata 7	karakterisstiki	karakteristik
8.	Kata 8	pemproswsan	pemrosesan
9.	Kata 9	promossi	promosi
10.	Kata 10	sellulra	seluler

Tabel 5. Kesalahan Penulisan karena Penggantian Huruf

No	Kata ke-	Kata Salah	Kata yang benar
1.	Kata 1	psikplpgi	psikologi
2.	Kata 2	memmrlukan	memerlukan
3.	Kata 3	sakah	salah
4.	Kata 4	manuak	manual
5.	Kata 5	imformasi	informasi
6.	Kata 6	computer	komputer
7.	Kata 7	pwngelolaan	pengelolaan
8.	Kata 8	mwlakykan	melakukan
9.	Kata 9	serungkalo	seringkali
10.	Kata 10	metods	metode

Tabel 6. Kesalahan Penulisan karena Penukaran Huruf

No	Kata ke-	Kata Salah	Kata yang benar
1.	Kata 1	kosnultasi	konsultasi
2.	Kata 2	yagn	yang
3.	Kata 3	keterlabmatan	keterlambatan
4.	Kata 4	baragn	barang
5.	Kata 5	dokumentsai	dokumentasi
6.	Kata 6	kemabli	kembali
7.	Kata 7	teresbut	tersebut
8.	Kata 8	daapt	dapat
9.	Kata 9	berorientsai	berorientasi
10.	Kata 10	prelu	perlu

4.2. Skenario Eksperimen

Pada studi ini dilakukan dua skenario eksperimen yang berbeda. Eksperimen satu dilakukan dengan cara mengujicobakan 50 kata yang salah sebagai data masukan untuk proses identifikasi kesalahan pengetikan dan saran perbaikannya berdasarkan algoritma *Hamming Distance, Levenshtein Distance, Damerau Levenshtein Distance* dan *Jaro Winkler Distance*. Selanjutnya dihitung nilai MAP untuk skenario uji coba tersebut.

Eksperimen kedua dilakukan menggunakan lingkungan yang sama, yaitu 50 kata yang salah dengan kesalahan pengetikan yang terbagi ke dalam empat jenis, yaitu kesalahan penulisan karena penghapusan huruf, penambahan huruf, penggantian huruf dan penukaran huruf. Dari 50 kata salah terbagi menjadi 12 kata pada jenis kesalahan pertama, 12 kata pada jenis kesalahan kedua, 13 kata pada jenis kesalahan ketiga dan 13 kata pada jenis kesalahan keempat. Tiap jenis kesalahan diproses dengan mencocokkan kata di dalam kamus berdasarkan algoritma *Hamming Distance, Levenshtein Distance, Damerau Levenshtein Distance* dan *Jaro Winkler Distance.*

4.3. Hasil dan Analisa

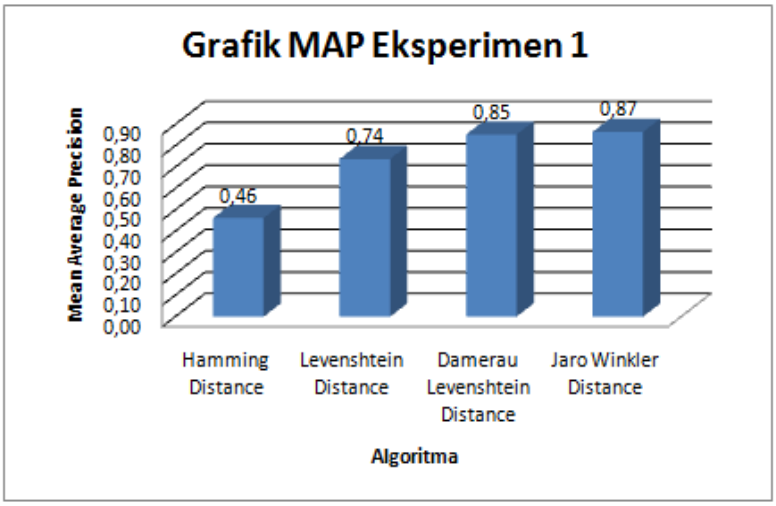
Hasil dari kedua skenario eksperimen yang telah dilakukan kemudian diteliti dan dianalisa. (a) Hasil Skenario Eksperimen 1 dan Analisa. Berdasarkan 50 kata salah yang diproses menggunakan empat algoritma menghasilkan nilai MAP masing-masing. Ada dua *user relevance judgement* yang terlibat untuk menentukan kata saran yang muncul relevan atau tidak. Hasil eksperimen 1 dapat dilihat pada Tabel 7 dan grafik MAP hasil eksperimen 1 dapat dilihat pada Gambar 4. Dari Gambar 4 dapat dilihat bahwa nilai MAP tertinggi terdapat pada algoritma *Jaro Winkler Distance* dengan nilai 0,87 sedangkan yang terendah adalah algoritma *Hamming Distance* dengan nilai 0,46. Berdasarkan grafik tersebut dapat diketahui bahwa algoritma terbaik untuk eksperimen 1 dengan penggunaan 50 kata salah dengan kesalahan yang tidak ditentukan adalah algoritma *Jaro Winkler Distance*. (b) Hasil Skenario Eksperimen 2. Berdasarkan empat jenis koleksi kata yang salah maka dapat diimplementasikan ke dalam grafik dengan tabel grafik seperti pada Tabel 8 dan grafik MAP hasil eksperimen 2 dapat dilihat pada Gambar 5.

Tabel 7. Hasil Eksperimen 1

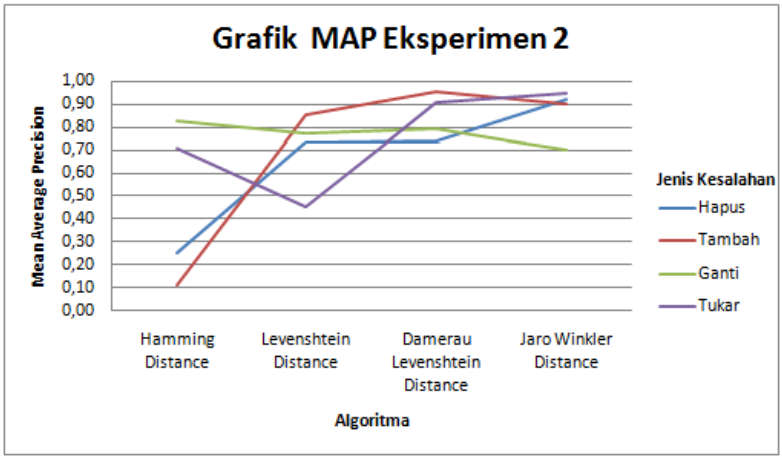
Algoritma	MAP
Hamming Distance	0,46
Levenshtein Distance	0,74
Damerau Levenshtein Distance	0,85
Jaro Winkler Distance	0,87

Tabel 8. Hasil Eksperimen 2

Algoritma	Jenis Kesalahan			
	Hapus	Tambah	Ganti	Tukar
Hamming Distance	0,25	0,11	0,83	0,71
Levenshtein Distance	0,73	0,85	0,78	0,46
Damerau Levenshtein Distance	0,74	0,95	0,80	0,91
Jaro Winkler Distance	0,92	0,90	0,70	0,95



Gambar 4. Grafik Eksperimen 1



Gambar 5. Grafik Eksperimen 2

Grafik di atas menunjukkan bahwa untuk kesalahan penulisan karena penghapusan huruf, algoritma yang baik adalah *Jaro Winkler Distance* karena memiliki nilai MAP tertinggi yaitu 0,92. Pada kesalahan penulisan karena penambahan huruf, algoritma yang baik adalah *Damerau Levenshtein Distance* karena memiliki nilai MAP tertinggi yaitu 0,95. Pada kesalahan penulisan karena penggantian huruf, algoritma yang baik adalah *Hamming Distance* karena memiliki nilai MAP tertinggi yaitu 0,83. Pada kesalahan penulisan karena penukaran huruf, algoritma yang baik adalah *Jaro Winkler Distance* karena memiliki nilai MAP tertinggi yaitu 0,95.

5. Kesimpulan dan Saran

5.1. Kesimpulan

Kesimpulan yang dapat diambil dari penelitian mengenai Perbandingan Algoritma Pencarian *String* dalam Metode *Approximate String Matching* untuk Identifikasi Kesalahan Pengetikan Teks Bahasa Indonesia dan Saran Perbaikan adalah perbandingan algoritma pencarian *string* dalam Metode *Approximate String Matching* pada kedua eksperimen menunjukkan bahwa algoritma *Jaro Winkler Distance* memiliki nilai tertinggi dibandingkan ketiga algoritma yang lain dengan nilai MAP 0,87 yang terbagi ke dalam empat jenis kesalahan penulisan yaitu jenis kesalahan penghapusan huruf 0,92, jenis kesalahan penambahan huruf 0,90, jenis kesalahan penggantian huruf 0,70 dan jenis kesalahan penukaran huruf 0,95.

5.2. Saran

Beberapa saran yang dapat diberikan untuk penelitian selanjutnya adalah sebagai berikut: (1) Pengecekan ini masih sebatas pengecekan kesalahan pengetikan, sehingga masih dapat dikembangkan untuk pengecekan dengan menggunakan pola kalimat berbahasa Indonesia. (2) Penelitian ini dapat dikembangkan untuk pengecekan dokumen dalam jumlah besar. (3) Kamus yang digunakan dapat dikembangkan agar dapat di-*update* sehingga perbendaharaan kata yang ada semakin banyak. (4) Penelitian ini dapat dikembangkan untuk penggantian saran kata secara otomatis yang langsung menggantikan kata yang salah.

Referensi

Hamming, R.W., 1950. Error detecting and error correcting codes. *Bell System Technical Journal*, XXIX(2), pp.147-60.

Khatami, S., 2013. Comparison and Improvement of Basic String Metrics for Surname Matching. *Life Science Journal*, X(5), pp.128-32.

Kornain, A., Yansen, F., & Tinaliah, T. 2014. Penerapan Algoritma Jaro-Winkler Distance untuk Sistem Pendeteksi Plagiarisme pada Dokumen Teks Berbahasa Indonesia. *Jurnal Ilmu Komputer dan Teknologi Informasi*, pp.62-70.

Luqman, 2009. Perancangan Program Aplikasi Pengoreksian Ejaan Bahasa Indonesia Berbasis Web. *Jurnal Pengkajian dan Penerapan Teknik Informatika (JURNAL PETIR)*, II(1), pp.13-19.

Mulyanto, A., 2010. Analisis Edit Distance Menggunakan Algoritma Dynamic Programming. *Saintek*, V(2).

Pyriana, D.R., 2011. *Program Aplikasi Editor Kata Bahasa Indonesia Menggunakan Metode Approximate String Matching dengan Algoritma Levenshtein Distance Berbasis Java*. Skripsi. Malang: Universitas Brawijaya.

Sagita, V. & Prasetiyowati, M.I., 2012. Studi Perbandingan Implementasi Algoritma Boyer-Moore, Turbo Boyer-Moore, dan Tuned Boyer-Moore dalam Pencarian String. *Ultimatics*, IV(1), pp.31-37.

Setiadi, I., 2013. *Damerau-Levenshtein Algorithm and Bayes Theorem for Spell Checker Optimization*. Bandung: Institut Teknologi Bandung.

Sutisna, U. & Adisantoso, J., 2010. Koreksi Ejaan Query Bahasa Indonesia Menggunakan Algoritma Damerau Levenshtein. *Jurnal Ilmiah Ilmu Komputer*, XV(2), pp.25-29.