 ISSN: 2684-8473

Penghapusan kolom dan baris pertama pada matriks distance untuk optimasi spell checker damerau-levenshtein distance

Puji Santoso[1], Pundhi Yuliawati[2], Ridwan Shalahuddin[3] , Ilham Ari Elbaith Zaeni[4]

Jurusan Teknik Elektro, Universitas Negeri Malang, Malang, Indonesia
[1] pujiosotnas@gmail.com; [2] pe.yhulia@gmail.com; [3] ridhwan102@gmail.com; [4] ilham.ari.ft@um.ac.id

INFORMASI ARTIKEL

Histori Artikel
Diterima : 22 Februari 2020
Direvisi : 2 Maret 2020
Diterbitkan : 4 April 2020

Kata Kunci:
Pengecekan ejaan
Levensthein distance
Damerau-levensthein distance
Optimasi

ABSTRAK

Damerau-Levenshtein Distance menentukan jarak atau jumlah minimum operasi yang dibutuhkan untuk mengubah satu string menjadi string lain, di mana operasi yang digunakan untuk menentukan tingkat kemiripian antar String adalah insertion, deletion, substitution dan transposition. Algoritma ini sendiri dapat juga digunakan untuk mengoreksi kesalahan kata. Namun, Algoritma Damerau-Levenshtein Distance mempunyai kelemahan, yaitu waktu pemrosesan yang lama. Pada perhitungan jarak antara dua string dengan algoritma Damerau-Levenshtein, setiap huruf dari kedua string akan dibandingkan dengan membuat matriks distance. Karena Kamus Bahasa Indonesia memiliki lebih dari 30.000 kata dasar, operasi perhitungan jarak akan dilakukan lebih dari 30.000 kali untuk setiap kesalahan. Penelitian ini mengusulkan peningkatan untuk mempersingkat waktu pemrosesan algoritma Damerau-Levenshtein dengan mengurangi baris dan kolom matriks distance. Hasil akhir yang diharapkan dari penelitian ini adalah waktu pemrosesan menjadi lebih cepat tanpa harus mengorbankan akurasi.

I. Pendahuluan

Levensthein Distance adalah salah satu algoritma untuk pengecekan ejaan dengan tingkat akurasi yang cukup baik [1], [2]. Algoritma ini mampu menghitung minimum operasi yang dibutuhkan untuk mengubah suatu kata menjadi kata lain atau disebut sebagai jarak *edit* [3]. Jenis operasi yang dihitung hanya terbatas pada operasi *insertion* (penambahan karakter), *deletion* (penghapusan karakter), dan *subtitution* (penggantian karakter) [4]. Levenshtein Distance kemudian dikembangkan menjadi Damerau Lavenshtein Distance dengan menambahkan perhitungan untuk proses *transposisi* (penukaran posisi dua karakter) seperti kata “aku” dan “kau” [5].

Damerau Levenshtein Distance meningkatkan akurasi Levenshtein [6]. Namun, algoritma ini tidak memperbaiki kompleksitas waktu pemrosesan Levenshtein Distance terutama saat menggunakan kamus dengan ukuran besar [7]. Agar sistem dapat menampilkan saran kata pada kata yang salah, sistem harus melakukan perhitungan pada kata tersebut dengan setiap kata di dalam kamus. Untuk setiap kata yang dibandingkan, sistem akan membuat sebuah matriks distance berukuran $m_{+1} \times n_{+1}$ dimana m adalah jumlah huruf pada kata pertama dan n adalah jumlah huruf pada kata kedua [8]. Pembuatan matriks untuk setiap kata di dalam kamus besar tentunya akan membutuhkan banyak waktu.

Beberapa penelitian meningkatkan algoritma Levenshtein agar dapat diandalkan untuk tujuan tertentu. Sebagai contoh, Pal dan Rajasekaran meningkatkan algoritma Levenshtein untuk menemukan motif dalam serangkaian string biologis [9]. Peningkatan ini membuat algoritma Levenshtein cocok untuk menemukan motif dalam string biologis. Namun, ini menjadikan kinerja algoritma lebih lambat dari aslinya. Navarro, dkk dan Camarena, dkk meningkatkan algoritma Levenshtein menjadi lebih cepat untuk pengambilan informasi musik dengan mengabaikan proses *deletion* dan *insertion* dari perhitungan [10], [11]. Namun, ketiga penelitian tersebut hanya bisa digunakan untuk satu tujuan khusus.

Mempercepat waktu proses tanpa mengurangi akurasi dapat diraih dengan menghapus baris dan kolom pertama pada matriks distance yang digunakan. Pengurangan ukuran matriks distance terbukti mengurangi biaya komputasi untuk penyimpanan dan mempersingkat waktu pemrosesan Levenshtein Distance karena proses pembuatan baris dan kolom pertama akan dilewatkan [12]. Pada penilitan ini, penghapusan baris dan kolom pertama akan diterapkan pada Algoritma Damerau Levenshtein Distance (DLD).

II. Metode

A. Pengumpulan Data

Dataset yang digunakan yaitu dua dongeng dan diambil dari situs *ceritadongengrakyat.com*. Kedua dongeng tersebut memiliki total 1266 kata yang akan digunakan sebagai data uji. Dongeng atau cerita pendek biasanya ditulis dengan menggunakan aplikasi pengolah kata, tetapi biasanya terdapat kesalahan ejaan pada penulisan. Kesalahan ejaan terjadi jika kata yang dituliskan oleh penulis tidak ada atau tidak terdaftar pada KBBI (Kamus Besar Bahasa Indonesia). Kesalahan ejaan disebabkan oleh beberapa hal, seperti ketidaktahuan dalam penulisan, kesalahan pada saat penulisan atau pengetikan dan kesalahan pada mesin pada saat penyimpanan. Beberapa contoh kesalahan yang sering terjadi adalah sebagai berikut:

- Kesalaan pada penggantian satu huruf seperti kata '*Aku*' menjadi '*Aju*'.
- Kesalahan dengan penyisipan satu huruf seperti kata '*Aku*' menjadi '*Akku*'.
- Kesalahan dengan menghilangkan satu huruf seperti kata '*Aku*' menjadi 'A*u*'.
- Kesalahan pada penukaran dua huruf berdekatan seperti kata '*Aku*' menjadi '*Auk*'.
- Kesalahan pada dua kata yang tidak diberi spasi seperti kata '*Aku Ingin*' menjadi '*AkuIngin*'.

Untuk mengukur waktu pemrosesan, data uji dipisahkan menjadi sepuluh bagian dengan jumlah kalimat yang berbeda. Terdapat data uji yang hanya berisi satu kalimat, hingga data uji yang berisi satu paragraf. Jumlah kata dalam data uji yang paling sedikit adalah 20 kata, sedangkan jumlah kata terbanyak adalah 464 kata.

Tiap data kemudian akan diuji dengan menggunakan Damerau Levenshtein Distance biasa dan dengan Optimasi Damerau Levenshtein Distance. Untuk mengetahui perbedaan kecepatan waktu proses, tiap percobaan akan diberi program untuk mengukur waktu proses.

B. Damerau Levenshtein Distance (DLD)

Perhitungan pada DLD dilakukan dengan menghitung jarak *edit* menggunakan matriks distance. Gambar 1 menunjukkan matriks distance untuk kata DMERAU dan DAMERAU. Jarak *edit* kedua kata tersebut adalah 1, didapatkan dari sel terakhir.

C. Optimasi DLD (O-DLD)

Proses optimasi yang dilakukan adalah menghapus baris dan kolom pertama pada matriks distance. Untuk lebih jelasnya, dapat dilihat perbedaan antara Gambar 1 dan Gambar 2.

		D	M	E	R	A	U
	0	1	2	3	4	5	6
D	1	0	1	2	3	4	5
A	2	1	1	2	3	3	4
M	3	2	1	2	3	4	4
E	4	3	2	1	2	3	4
R	5	4	3	2	1	2	3
A	6	5	4	3	2	1	2
U	7	6	5	4	3	2	1

Gambar. 1. Matriks distance pada DLD

	D	M	E	R	A	U
D	0	1	2	3	4	5
A	1	1	2	3	3	4
M	2	1	2	3	4	4
E	3	2	1	2	3	4
R	4	3	2	1	2	3
A	5	4	3	2	1	2
U	6	5	4	3	2	1

Gambar. 2. Matriks distance pada O-DLD

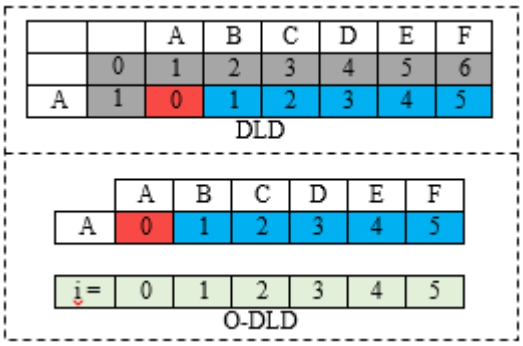
Pembuatan baris dan kolom pertama pada DLD dilakukan dengan fungsi sederhana. Perhatikan pada Gambar 1, baris pertama bernilai 0, 1, 2, .., 6 dimana 6 adalah panjang kata DMERAU dan kolom pertama bernilai 0, 1, 2, .., 7 dimana 7 adalah panjang kata DAMERAU. Kedua pola tersebut dapat disebut sebagai *i* dan *j*. Dengan menggunakan *i* dan *j* sebagai pengganti baris dan kolom pertama, pembuatan matriks distance dapat dilakukan. Matriks dapat dibuat dengan tiga langkah yaitu (a) pembuatan sel pertama atau d(0,0), (b) pembuatan sel pada baris pertama atau d(0, i), dan (c) pembuatan sel pada kolom pertama atau d(j, 0).

Langkah pertama yaitu pembuatan sel pertama atau d(0,0). Pada Gambar 3 dan Gambar 4, sel berwarna merah dilakukan dengan peraturan sebagai berikut:

- Sel pertama akan bernilai 0 jika huruf pertama pada kedua kata sama,
- Sel pertama akan bernilai 1 jika huruf pertama pada kedua kata berbeda.

Langkah kedua yaitu pembuatan sel pada baris pertama atau *d*(0, *i*). Ppada Gambar 3, sel berwarna biru dilakukan dengan peraturan berikut:

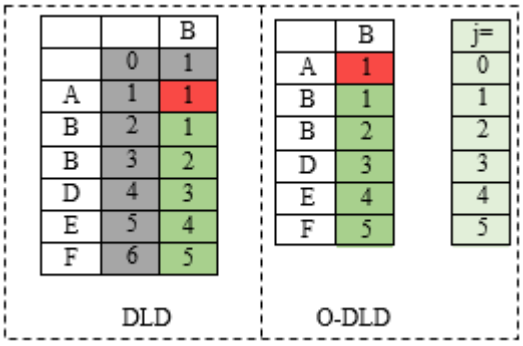
- Jika kedua huruf yang dibandingkan sama, sel akan bernilai i,
- Jika berbeda, sel akan mengambil nilai terkecil antara i + 1, atau d(0, i-1) + 1 atau nilai sel dikirinya ditambah 1.



Gambar. 3. Baris Pertama DLD dan O-DLD

Langkah ketiga yaitu pembuatan sel pada kolom pertama atau *d*(*j*, 0). Pada Gambar 4, sel berwarna hijau dilakukan dengan peraturan berikut:

- Jika kedua huruf yang dibandingkan sama, sel akan bernilai j,
- Jika berbeda, sel akan mengambil nilai terkecil antara j + 1, atau d(j-1, 0) + 1 atau nilai sel diatasnya ditambah 1.



Gambar. 4. Kolom Pertama DLD dan O-DLD

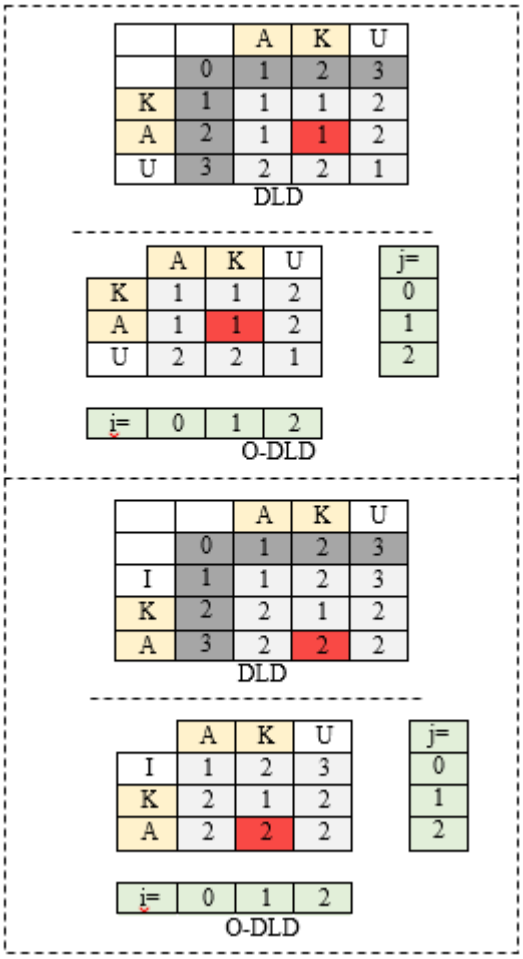
Pada DLD terdapat operasi transposisi untuk menghitung pertukaran huruf dari kedua kata seperti kata ‘AKU’ dan ‘AUK’. Terdapat empat kondisi yang dapat terjadi, yaitu:

- Transposisi terjadi pada baris kedua
- Transposisi terjadi pada kolom kedua
- Transposisi terjadi pada baris dan kolom kedua
- Transposisi terjadi setelah baris dan kolom kedua

Kondisi pertama, kedua dan ketiga membutuhkan nilai baris dan kolom pertama dari matriks distance, sedangkan kondisi keempat dapat dilakukan tanpa nilai baris dan kolom pertama. Transposisi yang terjadi pada kondisi pertama, kedua dan ketiga dapat dilakukan dengan pertaturan sebagai berikut:

- Jika transposisi terjadi pada baris kedua, maka nilai transposisi = *i*
- Jika transposisi terjadi pada kolom kedua, maka nilai transposisi = *j*
- Jika transposisi terjadi pada baris dan kolom kedua, nilai *i* dan *j* akan sama, maka nilai transposisi = *i* atau *j*

Pada Gambar 5, transposisi ditandai dengan sel berwarna merah. Pada contoh pertama, transposisi terjadi pada baris dan kolom kedua. Pada contoh kedua transposisi terjadi pada kolom kedua.



Gambar. 5. Transposisi pada DLD dan O-DLD

D. Saran Kata dan Akurasi

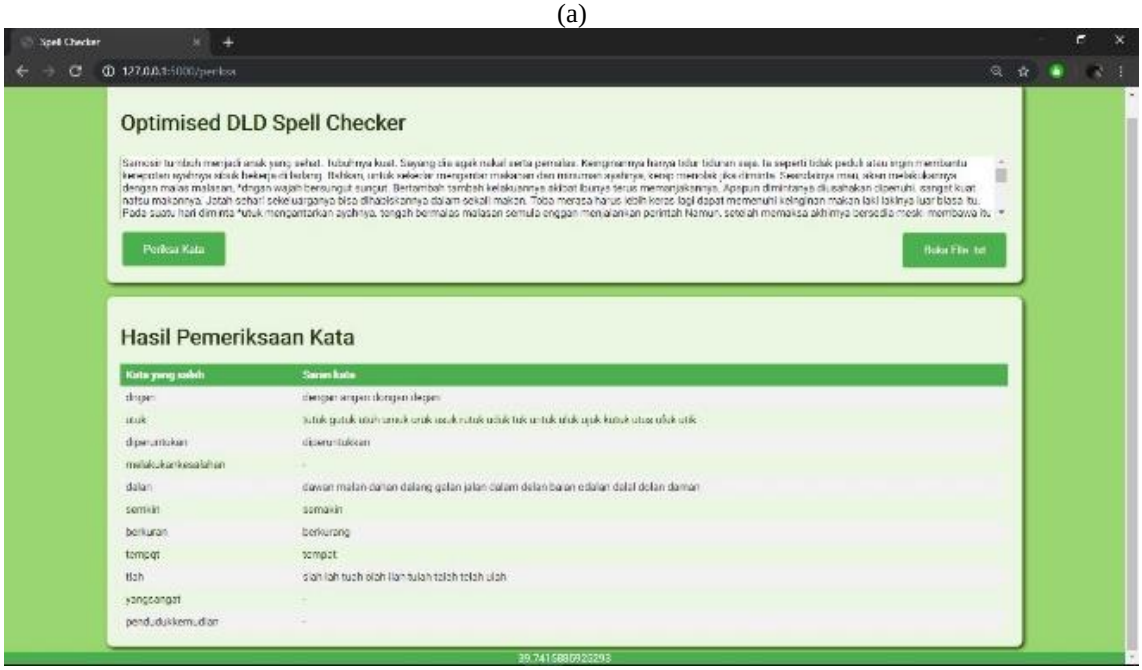
Saran kata adalah hasil proses pengujian berupa koreksi kesalahan kata. Saran kata yang dihasilkan adalah kata di kamus KBBI yang paling mirip dengan kata yang salah. Perhitungan akurasi dilakukan dengan menghitung jumlah kata salah yang dapat dikoreksi kemudian dibagi dengan jumlah kata yang salah dikali 100%. Rumus tersebut dapat dilihat pada persamaan (1), dimana x adalah jumlah kata salah yang dapat dikoreksi dan n adalah jumlah kata salah.

$$Akurasi = \frac{x}{n} X\ 100\% \quad (1)$$

III. Hasil dan Pembahasan

Pengujian dilakukan terhadap kedua dongeng dengan total 1622 kata. Berdasarkan hasil pengujian, didapatkan total 100 kesalahan pada dongeng. Hasil pengujian menunjukkan bahwa DLD maupun O-DLD tidak mampu memeriksa kesalahan kata yang tidak diberi spasi. Pengujian dengan O-DLD membutuhkan waktu proses yang lebih sedikit dari DLD. Perbandingan waktu proses dan akurasi DLD dan O-DLD dapat dilihat pada Gambar 6.

(a)



(b)

Gambar. 6. Hasil DLD (a) dan O-DLD (b)

Pada Gambar 6, terdapat tiga kesalahan kata tidak diberi spasi yang tidak dapat dikoreksi oleh DLD maupun O-DLD yaitu kata ‘melakukankesalahan’, ‘yangsangat’, dan ‘pendudukkemudian’. Sedangkan kesalahan lainnya mampu dikoreksi oleh kedua algoritma dengan hasil saran kata yang sama. Dengan demikian dapat dikatakan bahwa akurasi kedua algoritma tetap sama. Waktu proses dari kedua algoritma dapat dilihat pada bagian *footer* hasil. Waktu yang dibutuhkan oleh DLD adalah 72.59 detik sedangkan O-DLD hanya 39.74 detik.

Berdasarkan hasil pengujian, sebanyak 25 kesalahan kata dari total 100 kesalahan kata tidak mampu dikoreksi oleh kedua algoritma. Jumlah kesalahan kata yang berhasil dikoreksi oleh kedua metde adalah 75 kesalahan. Semua kesalahan yang tidak dapat dikoreksi adalah kesalahan kata tanpa spasi. Beberapa contoh kata tanpa spasi yang tidak dapat dikoreksi dapat dilihat pada Tabel 1.

Tabel 1. *Contoh Kesalahan Tidak Terdeteksi*

Kesalahan Kata	Saran Perbaikan Seharusnya
Putriraja	putri raja
suatuhari	suatu hari
adalahtitisan	adalah titisan
yangsangat	yang sangat
denganmarah	dengan marah
sangtkesal	sangat kesal
bekaslukaa	bekas luka
makhlukghaib	makhluk gaib

Tabel 2. *Waktu Proses DLD dan O-DLD (detik)*

No Data.	Jumlah Huruf	Huruf Salah	DLD	O-DLD
1	132	45	33.18	18.20
2	269	57	44.21	23.11
3	254	48	39.37	23.26
4	403	64	49.43	32.42
5	523	81	51.85	29.87
6	496	70	49.48	26.54
7	413	83	55.59	30.47
8	775	172	118.06	63.84
9	2.683	82	60.74	32.08
10	3.392	110	72.59	39.74

Waktu proses dari kedua algoritma memiliki perbedaan tergantung dengan jumlah huruf pada data uji dan jumlah huruf pada kata yang salah. Pada data uji dengan jumlah huruf dibawah 2000 huruf, rata-rata waktu proses yang dibutuhkan DLD adalah 55.15 detik, sedangkan O-DLD adalah 30.96 detik. Data uji dengan jumlah huruf diatas 2000 huruf, DLD membutuhkan waktu proses 66.67 detik, dan O-DLD membutuhkan waktu proses 35.91 detik. Waktu proses terlama dari kedua metode didapat dari jumlah huruf salah terbanyak, yaitu 172 huruf. Lebih rinci, hasil pengujian dapat dilihat pada Tabel 2.

IV. Kesimpulan

Berdasarkan hasil pengujian, kedua algoritma menghasilkan nilai akurasi yang sama yaitu 75%. Waktu proses yang dibutuhkan kedua algoritma sangat dipengaruhi oleh jumlah huruf pada kata yang salah. Semakin banyak huruf yang diperiksa, semakin lama pula waktu proses yang dibutuhkan. Jumlah huruf yang diperiksa juga berpengaruh pada waktu proses. Berdasarkan data uji, rata-rata waktu proses yang dibutuhkan oleh DLD adalah 57.45 detik, sedangkan O-DLD membutuhkan 31.95 detik. Berdasarkan hasil tersebut, waktu proses O-DLD lebih cepat 25.5 detik dari DLD.

Daftar Pustaka

[1] F. Ahmad and G. Kondrak, "Learning a spelling error model from search query logs," in *Proceedings of Human Language Technology Conference and Conference on Empirical Methods in Natural Language Processing*, 2005, pp. 955–962.

[2] M. Serva and F. Petroni, "Indo-European languages tree by Levenshtein distance," *EPL (Europhysics Lett.*, vol. 81, no. 6, 2008.

[3] Z. Su, B. Ahn, K. Eom, M. Kang, J. Kim, and M. Kim, "Plagiarism Detection Using the Levenshtein Distance and Smith-Waterman Algorithm," pp. 0–3, 2008.

[4] M. Nejja and A. Yousfi, "The Context in Automatic Spell Correction," *Procedia Comput. Sci.*, vol. 73, pp. 109–114, 2015.

[5] E. Y. and O. M. R. Gabrys, "Codes in the Damerau Distance for Deletion and Adjacent Transposition Correction," *IEEE Trans. Inf. Theory*, vol. 64, n, no. 4, 2018.

[6] A. S. Lhoussain, G. Hicham, and Y. Abdellah, "Adaptating the Levenshtein Distance to Contextual Spelling Correction," vol. Vol. 12, N, no. March, 2015.

[7] Y. J. and M. A. S. K. Balhaf, M. A. Alsmirat, M. Al-Ayyoub, "Accelerating Levenshtein and Damerau Edit Distance Algorithms Using GPU With Unified Memory," 2017.

[8] G. V Bard, "Spelling-Error Tolerant , Order-Independent Pass-Phrases via the Damerau-Levenshtein String-Edit Distance Metric," 2005.

[9] S. Pal and S. Rajasekaran, "Improved algorithms for finding edit distance based motifs," in *2015 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*, 2015, pp. 537–542.

[10] G. Navarro and M. Raffinot, *Flexible Pattern Matching in Strings. Practical On-Line Search Algorithms for Texts and Biological Sequences*, 17th ed. 2002.

[11] A. Camarena-Ibarrola and E. Ch, "On Musical Performances Identification , Entropy and String Matching On Musical Performances Identification ," no. November, 2006.

[12] H. N. Abdulkhudhur, I. Q. Habeeb, Y. Yusof, and S. A. M. Yusof, "Implementation Of Improved Levenshtein Algorithm For Spelling Correction Word," vol. 88, no. 3, pp. 6–9, 2016.