PEMBOBOTAN DINAMIS BERBASIS POSISI PADA *APPROXIMATE STRING MATCHING*

Sebastianus A. S. Mola[1], Meiton Boru[2] dan Emerensye S. Y. Pandie[3]

[1,2,3]Program Studi Ilmu Komputer, Fakultas Sains dan Teknik, Universitasn Nusa Cendana, Indonesia
[1]Email: adimola@staf.undana.ac.id
[2]Email: meitonboru@staf.undana.ac.id
[3]Email: emerensyepandie@staf.undana.ac.id

ABSTRAK

Komunikasi tertulis dalam media sosial yang menekankan pada kecepatan penyebaran informasi sering kali terjadi fenomena penggunaan bahasa yang tidak baku baik pada level kalimat, klausa, frasa maupun kata. Sebagai sebuah sumber data, media sosial dengan fenomena ini memberikan tantangan dalam proses ekstraksi informasi. Normalisasi bahasa yang tidak baku menjadi bahasa baku dimulai pada proses normalisasi kata di mana kata yang tidak baku (*non-standard word* (NSW)) dinormalisasikan ke bentuk baku (*standard word* (SW)). Proses normalisasi dengan menggunakan *edit distance* memiliki keterbatasan dalam proses pembobotan nilai *mismatch*, *match*, dan *gap* yang bersifat statis. Dalam perhitungan nilai *mismatch*, pembobotan statida tidak dapat memberikan pembedaan bobot akibat kesalahan penekanan tombol pada *keyboard* terutama tombol yang berdekatan. Karena keterbatasan pembobotan *edit distance* ini maka dalam penelitian ini diusulkan sebuah metode pembobotan dinamis untuk bobot *mismatch*. Hasil dari penelitian ini adalah adanya metode baru dalam pembobotan dinamis berbasis posisi tombol *keyboard* yang dapat digunakan dalam melakukan normalisasi NSW menggunakan metode *approximate string matching*.
Kata kunci: pembobotan dinamis, bobot *mismatch*, normalisasi NSW, *approximate string matching*

ABSTRACT

Written communication in social media that emphasizes the speed of information dissemination, the phenomenon of using non-standard language often occurs at the level of sentences, clauses, phrases and words. As a source of data, social media with this phenomenon presents challenges in the process of extracting information. Normalization of non-standard language into standard language begins in the word normalization process where non-standard words (NSW) are normalized to standard forms (standard words (SW)). The normalization process using edit distance has limitations in the process of weighting the static mismatch, match, and gap values. In calculating the mismatch value, statida weighting cannot provide a weight difference due to incorrect keystrokes on the keyboard, especially adjacent keys. Due to the limited edit distance weighting, this research proposes a dynamic weighting method for mismatch weights. The result of this research is that there is a new method of dynamic weighting based on the position of the keyboard keys that can be used to normalize NSW using the approximate string matching method.
Keywords: dynamic weighting, mismatch weight, NSW normalization, *approximate string matching*

1. PENDAHULUAN

Media sosial adalah *new media* berbasis komputer yang memfasilitasi dan mempermudah penggunanya dalam berekspresi, berinteraksi, dan mendapatkan informasi secara *online* (daring). Media sosial memudahkan seseorang atau penggunanya untuk membagikan ide, karya, pikirannya melalui komunitas yang terbangun secara online. Media sosial menggunakan teknologi berbasis *website* atau aplikasi, dengan bantuan internet dan perangkat seperti komputer ataupun *smartphone* untuk mengaksesnya [1].

Penggunaan NSW sering dijumpai dalam media sosial. Penelitian yang dilakukan oleh [2] menemukan bahwa penggunaan NSW dapat terjadi karena kesalahan pengetikan. Kesalahan pengetikan terjadi karena kegagalan menyesuaikan posisi jari dengan tombol-tombol pada *keyboard*. Bagi pengguna android yang memiliki jari berukuran besar sementara layar untuk menampilkan tombol *keyboard* cukup kecil, peluang untuk melakukan kesalahan pengetikan atau *typo* semakin besar. Contohnya kesalahan pengetikan antara huruf I dengan U, huruf A dengan S, atau antara huruf T dengan Y. Selain karena

ISSN: 2337-7631 (printed)
ISSN: 2654-4091 (Online)

kesalahan penggunaan NWS juga disebabkan oleh kesalahan ejaan seperti kata ’kalo’ untuk menyebutkan ’kalau’ [2], [3].

Tidak hanya pengetikan huruf, tanda baca juga kerap kali disepelekan demi alasan praktis, seperti tanda koma maupun titik. Selain itu, unsur kesengajaan sering kali digunakan untuk NSW, seperti ’tx’ untuk ’terima kasih’,’q’ untuk pengganti kata ’ak’, dan lain sebagainya. Ternyata, kesalahan kecil yang kerap disepelekan orang tersebut dapat berakibat fatal [4].

Usaha untuk memperbaiki kesalahan-kesalahan tersebut telah dilakukan khususnya dengan menggunakan *approximate string matching* seperti [5] yang menggunakan metode Needleman-Wunsch, [6], [7] dengan metode Jaro-Winkler, [7], [8] dengan metode Levenstein, [9] dengan metode *longest common sequence* (LCS), dan [10], [11] dengan menggunakan metode Smith-Waterman. Penelitian-penelitian tersebut ternyata menggunakan bobot tetap untuk menilai jarak antara dua huruf yang dibandingkan. Persoalan yang timbul dari pembobotan statis ini adalah lebih dimungkinkan adanya kesamaan bobot akhir dari beberapa kata yang mirip dengan NSW. Misalnya untuk NSW ’makun’ akan menghasilkan bobot akhir yang sama untuk kata ’makan’, dan ’makin’ walaupun secara umum diketahui bahwa huruf ’u’ pada kata ’makun’ terjadi akibat adanya kesalahan pengetikan huruf ’i’ dari kata ’makin’ karena posisi huruf ’u’ terhadap huruf ’i’ di *keyboard* lebih dekat dari posisi huruf ’u’ terhadap huruf ’a’.

Oleh karena adanya keterbatasan dalam pembobotan statis tersebut, penelitian ini akan mengusulkan sebuah pendekatan berbasis posisi untuk mendapatkan bobot dinamis *mismatch* yang dapat digunakan dalam semua metode *approximate string matching*.

2. MATERI DAN METODE

Approximate String Matching

Approximate string matching adalah teknik menemukan *string* yang paling mirip dengan pola *string* tertentu. Teknik ini melibatkan *edit distance* yakni jarak minimum dari pencocokan *string* yang meliputi proses pengubahan (*edit*) string seperti penyisipan (*insertion*), penghapusan (*deletion*) dan penggantian (*substitution*). Proses penyisipan dan penghapusan akan mengakibatkan adanya bobot *gap* sedangkan proses penggantian mengakibatkan adanya bobot *mismatch*. Bobot *match* terjadi jika tidak ada proses *edit* yang menandakan posisi tertentu dari kedua *string* sama. Persamaan *edit distance* secara umum seperti pada persamaan 1 [12]:

$$D(i,j) = min\begin{cases} D(0,j), & 0 \le j \le n \\ D(i-1,j)+1 \\ D(i-1,j-1)+\alpha & \alpha = \begin{cases} 0, & p_i = t_j \\ 1, & lainnya \end{cases} \\ D(i,j-1)+1 \end{cases} \quad \text{..... (1)}$$

dengan:

D(i,j) = *edit distance* minimum antara *string* p_1, p_2, …, p_i dengan *substring* T yang diakhiri dengan t_j

α = bobot *mismatch.*

Persamaan 1 menunjukan bahwa bobot *mismatch* (α) dalam *edit distance* selalu tetap yakni sebesar 1 jika kedua string pada posisi i-1 dan j-1 tidak sama.

Bobot *Mismatch* dalam Levenstein *Distance*

Persamaan *edit distance* dengan metode Levenstein ditunjukan pada persamaan 2 [13].

$$Lev_{p,t}(i,j) = \begin{cases} \max(i,j) \\ \begin{cases} Lev_{p,t}(i-1,j)+1 \\ Lev_{p,t}(i,j-1)+1 \\ Lev_{p,t}(i-1,j-1)+1_{(p_i \neq t_j)} \end{cases} \end{cases} \quad \text{..... (2)}$$

dengan:

$Lev_{p,t}(i,j) =$ *edit distance* minimum antara *string* p_1, p_2, …, p_i dengan *substring* T yang diakhiri dengan t_j.

Bobot *mismatch* yang digunakan dalam metode Levenstein sebesar 1 ketika kondisi $p_i \neq t_j$ terpenuhi.

Bobot *Mismatch* dalam Demearu-Levenstein *Distance*

Demearu-Levenstein *distance* merupakan perbaikan dari metode Levenstein. Namun, perbaikan pada metode Levenstein tidak terjadi pada bobot *mismatch*. Persamaan 3 menunjukkan bahwa bobot *mismatch* dalam metode Demerau-Levenstein ($c(p_i, t_j)$) masih berupa nilai konstanta sebesar 1.

ISSN:2337-7631 (printed)
ISSN: 2654-4091 (Online)

$$DL(i,j) = min\begin{cases} DL(i-1,j-1) + C(p_i,t_j), jika\ i,j > 0 \\ DL(i,j-1) + 1, jika\ i > 0 \\ DL(i-1,j) + 1, jika\ j > 0 \\ DL(i-2,j-2) + 1, if\ i,j > 0\ dan\ p_i = t_{j-1} dan\ p_{i-1} = t_j \end{cases} \quad \text{..... (3)}$$

Dengan $DL(i,j) = edit\ distance$ minimum antara *string* $p_1, p_2, \ldots, p_i$ dengan *substring* T yang diakhiri dengan t_j, $C(p_i,t_j)$ = bobot *mismatch* jika $p_i \neq t_j$.

Bobot *Mismatch* dalam Smith-Waterman

Metode Smith-Waterman merupakan metode pencocokan *string* yang memiliki nilai pencocokan seperti pada persamaan 4 [14]:

$$SW(i,j) = \begin{cases} SW(i-1,j-1) + s(p_i,t_j) \\ max_{k\geq 1}\{SW(i-k,j) - W_k\},\ (1 \leq i \leq n, 1 \leq j \leq m) \\ max_{l\geq 1}\{SW(i,j-l) - W_l\},\ (1 \leq i \leq n, 1 \leq j \leq m) \\ 0 \end{cases} \quad \text{..... (4)}$$

dengan:

$SW(i,j) = edit\ distance$ minimum antara *string* $p_1, p_2, \ldots, p_i$ dengan *substring* T yang diakhiri dengan t_j

$SW(i-1,j-1) + s(p_i,t_j)$ = *edit distance string* p_i dan t_j

$s(p_i,t_j)$ = bobot *mismatch* jika $p_i \neq t_j$,

$SW(i-k,j) - W_k$ = *edit distance* jika p_i berada di akhir gap dengan panjang k

$SW(i,j-l) - W_l$ = *edit distance* jika t_j berada di akhir gap dengan panjang l, 0 = tidak ada kecocokan antara *string* p_i dan t_j.

Dari persamaan 4 tersebut diketahui bahwa obot *mismatch* pada metode Smith-Waterman juga bernilai konstan yakni sebesar 0.

Bobot *Mismatch* dalam Needleman-Wunsch

Dalam metode Needleman-Wunsch, bobot *mismatch* juga konstan sebesar $s(p_i,t_j)$ seperti pada persamaan 5 [15]:

$$NW(i,j) = \begin{cases} NW(i-1,j-1) + s(p_i,t_j) \\ NW(i-1,j) - W_1 \\ NW(i,j-1) - W_1 \end{cases} \quad \text{..... (5)}$$

dengan:

$NW(i,j) = edit\ distance$ minimum antara *string* $p_1, p_2, \ldots, p_i$ dengan *substring* T yang diakhiri dengan t_j

$NW(i-1,j-1) + s(p_i,t_j)$ = *edit distance string* p_i dan t_j, $s(p_i,t_j)$ = bobot *mismatch* jika $p_i \neq t_j$,

W_1 = bobot *gap*.

3. HASIL DAN PEMBAHASAN

Metode Pembobotan Dimamis yang Diusulkan

Telah diketahui bahwa bobot *mismatch* konstan yang digunakan dalam *approximate string matchin*g memperbesar kemungkinan adanya hasil akhir *edit distance* sama untuk beberapa *string* yang dibandingkan. Karena itu di sini akan diusulkan sebuah metode pembobotan dinamis berdasarkan posisi tombol *keyboard*. Metode pembobotan ini dibagi ke dalam 2 tahap yakni tahap pengindeksan tombol *keyboard* dan tahap perhitungan bobot *mismatch*.

Langkah-langkah pada tahap pengindeksan tombol *keyboard* adalah:

1. Buat indeks r untuk baris dan indeks c untuk kolom pada *layout keyboard*.
2. Representasikan setiap tombol *keyboard* dengan indeks baris dan kolom (r,c). Setiap tombol *keyboard* harus berada dalam satu baris namun dalam banyak kolom yakni kolom awal (c_{awal}) dan kolom akhir (c_{akhir}) sehingga tiap tombol direpresentasikan dengan (r, (c_{awal}, c_{akhir})).
3. Hitung lebar kolom (W_c) untuk semua tombol *keyboard* sesuai persamaan 6.
$$W_c = (c_{akhir} - c_{awal}) + 1 \quad \text{..... (6)}$$
4. Hitung titik pusat kolom ($center_c$) semua tombol *keyboard* sesuai persamaan 7.
$$center_c = (c_{awal} + c_{akhir})/2 \quad \text{..... (7)}$$

Langkah-langkah pada tahap perhitungan bobot kemiripan adalah:

1. Misalkan terdapat 2 *string* P($p_1, p_2, \ldots, p_i$) dan *string* T($t_1, t_2, \ldots, t_j$)
2. Untuk semua *substring* p_i dan t_j di mana i=j dan $p_i \neq t_j$ dan i=j
   a. Hitung selisih baris antara p_i dan t_j sesuai persamaan 8.

ISSN:2337-7631 (printed)
ISSN: 2654-4091 (Online)

$$\Delta r = |r_{p_i} - r_{t_j}| \quad (8)$$

b. Hitung selisih kolom antara p_i dan t_j:

i. Jika $W_{p_i} = W_{t_j}$ maka sesilih kolom sesuai persamaan 9.

$$\Delta c = \frac{|center_{p_i} - center_{t_j}|}{2} \quad (9)$$

ii. Jika $W_{p_i} \neq W_{t_j}$

- Jika $c_{akhir_{pi}} + 1 \leq c_{awal_{tj}}$ (tombol *keyboard* p_i berada di sebelah kiri tombol *keyboard* t_j) maka selisih kolom (Δc) sesuai persamaan 10.

$$\Delta c = \left| c_{akhir_{pi}} - c_{awal_{tj}} \right| \quad (10)$$

- Jika $c_{awal_{pi}} - 1 \geq c_{akhir_{tj}}$ (tombol *keyboard* p_i berada di sebelah kanan tombol *keyboard* t_j) maka selisih kolom (Δc) sesuai persamaan 11.

$$\Delta c = \left| c_{awal_{pi}} - c_{akhir_{tj}} \right| \quad (11)$$

- Jika tidak maka, selisih kolom (Δc) sesuai persamaan 12.

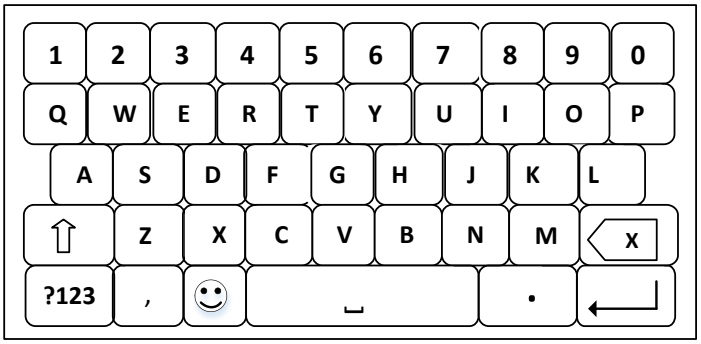
$$\Delta c = 0{,}5 \quad (12)$$

c. Hitung bobot *mismatch* (mw) sesuai persamaan 13.

$$mw = \Delta r + \Delta c \quad (13)$$

Ilustrasi Langkah-langkah Penindeksan

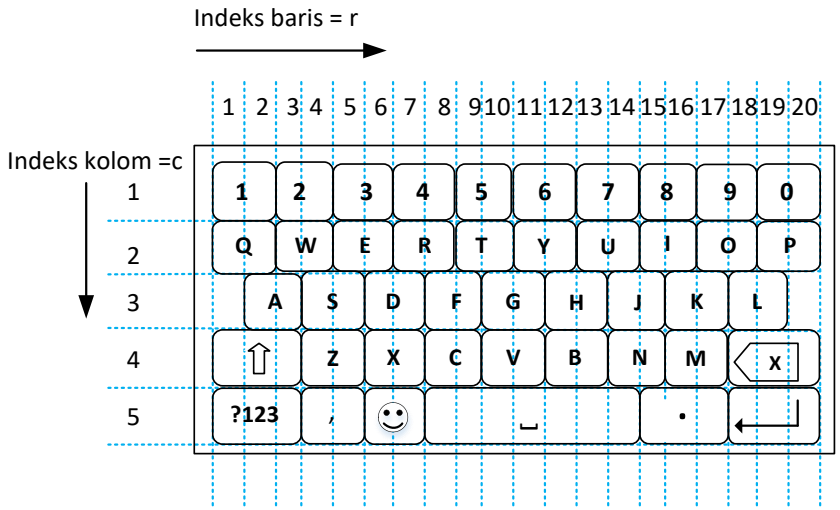
Ilustrasi langkah-langkah pengindeksan akan dimulai dengan mengambil salah satu *layout keyboard*. Misalkan terdapat *layout keyboard* seperti pada gambar 1. *Layout keyboard* pada gambar 1 adalah salah satu *layout keyboard* dari perangkat *handphone* dengan merk OPPO seri A92.



Gambar 1. *Layout keyboard* OPPO A92

Langkah-langkah pengindeksan tombol *keyboard* diilustrasikan sebagai berikut:

1. Buat indeks r untuk baris dan indeks c untuk kolom pada *layout keyboard* seperti pada gambar 2.



Gambar 2.*Layout keyboard* setelah diberi indeks baris dan kolom

2. Representasikan setiap tombol *keyboard* dengan indeks baris dan kolom. Hasil representasi setiap tombol ditunjukan pada tabel 1.

ISSN:2337-7631 (printed)
ISSN: 2654-4091 (Online)

3. Hitung lebar kolom (W_c) sesuai persamaan 6. Hasil perhitungannya dapat dilihat pada tabel 1.
4. Hitung titik pusat kolom ($center_c$) sesuai persamaan 7 Hasil perhitungannya dapat dilihat pada tabel 1.

Tabel 1. Representasi tombol *keyboard* dalam indeks baris dan kolom

Tombol	Indeks baris (r)	Indeks kolom (c)		Representasi tombol (r, (c_{awal}, c_{akhir})	W_c	$center_c$
		Awal (c_{awal})	akhir (c_{akhir})			
1	1	1	2	(1,(1,2))	2	1.5
2	1	3	4	(1,(3,4))	2	3.5
3	1	5	6	(1,(5,6))	2	5.5
4	1	7	8	(1,(7,8))	2	7.5
5	1	9	10	(1,(9,10))	2	9.5
6	1	11	12	(1,(11,12))	2	11.5
7	1	13	14	(1,(13,14))	2	13.5
8	1	15	16	(1,(15,16))	2	15.5
9	1	17	18	(1,(17,18))	2	17.5
0	1	19	20	(1,(19,20))	2	19.5
A	3	2	3	(3,(2,3))	2	2.5
B	4	12	13	(4,(12,13))	2	12.5
C	4	8	9	(4,(8,9))	2	8.5
D	3	6	7	(3,(6,7))	2	6.5
E	2	5	6	(2,(5,6))	2	5.5
F	3	8	9	(3,(8,9))	2	8.5
G	3	10	11	(3,(10,11))	2	10.5
H	3	12	13	(3,(12,13))	2	12.5
I	2	15	16	(2,(15,16))	2	15.5
J	3	14	15	(3,(14,15))	2	14.5
K	3	16	17	(3,(16,17))	2	16.5
L	3	18	19	(3,(18,19))	2	18.5
M	4	16	17	(4,(16,17))	2	16.5
N	4	14	15	(4,(14,15))	2	14.5
O	2	17	18	(2,(17,18))	2	17.5
P	2	19	20	(2,(19,20))	2	19.5
Q	2	1	2	(2,(1,2))	2	1.5
R	2	7	8	(2,(7,8))	2	7.5
S	3	4	5	(3,(4,5))	2	4.5
T	2	9	10	(2,(9,10))	2	9.5
U	2	13	14	(2,(13,14))	2	13.5
V	4	10	11	(4,(10,11))	2	10.5
W	2	3	4	(2,(3,4))	2	3.5
X	4	6	7	(4,(6,7))	2	6.5
Y	2	11	12	(2,(11,12))	2	11.5
Z	4	4	5	(4,(4,5))	2	4.5
KOMA	5	4	6	(5,(4,6))	2	4.5
SPASI	5	8	15	(5,(8,15))	7	11,5
TITIK	5	15	17	(5,(15,17))	3	16

Catatan: Tombol *Shift, Backspace* dan *Carriage Return* tidak diindeks sebab bukan tombol yang mewakili karakter yang dapat dicetak. Tombol '*?123'* dan *Emoticon* tidak diindeks karena tombol tersebut mengarahkan pengguna pada *layout keyboard* yang lain.

Ilustrasi Langkah-langkah Perhitungan Bobot

Ilustrasi perhitungan bobot melibatkan *string* P dan *string* T.

1. Misalkan terdapat dua *string* yang akan dibandingkan yakni P = MAKAN dan T = MAKAM. Panjang *string* P dan T = 5. *String* P adalah *string* pada NSW sedangkan *string* Q adalah *string* SW.
2. Untuk semua *substring* p_i dan t_j di mana i=j dan $p_i \neq t_j$ (yang memenuhi kriteria ini adalah i = j = 5 dan p_i = 'N' dan t_j = 'M')
    a. Hitung selisih baris antara p_i dan t_j sesuai persamaan 8.

ISSN:2337-7631 (printed)
ISSN: 2654-4091 (Online)

$\Delta r = |r_{p_i} - r_{t_j}| = |4 - 4| = 0$

d. Hitung selisih kolom antara p_i dan t_j. Karena $W_{p_i} = W_{t_j} = 2$ maka sesilih kolom sesuai persamaan 11.

$$\Delta c = \frac{|center_{p_i} - center_{t_j}|}{2} = \frac{|14{,}5 - 16{,}5|}{2} = 1$$

e. Hitung bobot *mismatch* (*mw*) sesuai persamaan 13:

$mw = \Delta r + \Delta c = 0 + 1 = 1$.

Maka diperoleh bobot *mismatch* antara *string* 'N' dan 'M' dalam kata 'MAKAN' dan 'MAKAM' sebesar 1. Namun jika *string* 'MAKAN' dibandingkan dengan *string* 'MAKAR' maka perhitungan bobot *mismatch*-nya menjadi:

1. Misalkan terdapat dua *string* yang akan dibandingkan yakni P = MAKAN dan T = MAKAR. Panjang *string* P dan T = 5.
2. Untuk semua *substring* p_i dan t_j di mana i=j dan $p_i \neq t_j$ (yang memenuhi kriteria ini adalah i = j = 5 dan p_i = 'N' dan t_j = 'R')

b. Hitung selisih baris antara p_i dan t_j sesuai persamaan 8.

$\Delta r = |r_{p_i} - r_{t_j}| = |4 - 2| = 2$

f. Hitung selisih kolom antara p_i dan t_j. Karena $W_{p_i} = W_{t_j} = 2$ maka sesilih kolom sesuai persamaan 11.

$$\Delta c = \frac{|center_{p_i} - center_{t_j}|}{2} = \frac{|14{,}5 - 7{,}5|}{2} = 3{,}5$$

g. Hitung bobot *mismatch* (*mw*) sesuai persamaan 13:

$mw = \Delta r + \Delta c = 2 + 3{,}5 = 5{,}5$.

Maka diperoleh bobot *mismatch* antara *string* 'N' dan 'M' dalam kata 'MAKAN' dan 'MAKAM' sebesar 3,5. Dari kedua ilustrasi tersebut dapat dilihat bahwa *string* 'MAKAN' lebih dekat jaraknya dengan 'MAKAM' dibandingkan dengan 'MAKAR' dilihat dari bobot *mismatch*-nya.

Tabel Bobot *Missmatch*

Keunggulan dari metode yang diusulkan ini adalah kesederhanaan dalam komputasinya sehingga bobot *mismatch* untuk setiap *layout keyboard* dapat direpresentasikan dalam tabel. Hal ini sangat penting mengingat metode ini harus diterapkan dalam metode-metode *approximate string-matching* yang bersifat iteratif. Jika telah disajikan dalam bentuk tabel maka beban komputasi untuk mendapatkan bobot *mismatch* telah jauh berkurang. Tabel 2 menunjukan bobot *mismatch* untuk semua kombinasi tombol pada contoh *layout keyboard* pada gambar 1. Tabel 2 dapat dipandang sebagai tabel *look-up* yang memuat keseluruhan bobot *mismatch* pada *layout keyboard* tersebut. Jika terjadi kondisi *mismatch* dalam perhitungan *edit distance* maka bobotnya dapat dilihat pada tabel *look-up* pada baris dan kolom yang menunjukan karakter *mismatch* dan karakter yang seharusnya. Kolom 1 pada tabel 2 menunjukan karakter pada NSW dan baris 1 pada tabel 2 menunjukan karakter pada SW.

Tabel 2. Bobot *mismatch* semua kombinasi tombol

Tombol	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**0**	**A**	**B**	**C**	**D**	**…**	**TITIK**
1	0	1	2	3	4	5	6	7	8	9	2,5	8,5	6,5	4,5	…	17
2	1	0	1	2	3	4	5	6	7	8	2,5	7,5	5,5	3,5	…	15
3	2	1	0	1	2	3	4	5	6	7	3,5	6,5	4,5	2,5	…	13
4	3	2	1	0	1	2	3	4	5	6	4,5	5,5	3,5	2,5	…	11
5	4	3	2	1	0	1	2	3	4	5	5,5	4,5	3,5	3,5	…	9
6	5	4	3	2	1	0	1	2	3	4	4,5	0,5	1,5	4,5	…	7
7	6	5	4	3	2	1	0	1	2	3	5,5	0,5	2,5	3,5	…	5
8	7	6	5	4	3	2	1	0	1	2	6,5	1,5	3,5	4,5	…	5
9	8	7	6	5	4	3	2	1	0	1	7,5	2,5	4,5	5,5	…	7
0	9	8	7	6	5	4	3	2	1	0	8,5	3,5	5,5	8,5	…	9
A	2,5	2,5	1,5	2,5	3,5	4,5	5,5	6,5	7,5	3,5	0	6	4	2	…	14
B	8,5	7,5	6,5	5,5	4,5	3,5	3,5	4,5	5,5	6,5	6	0	2	4	…	3
C	6,5	5,5	4,5	3,5	3,5	4,5	5,5	6,5	7,5	8,5	4	2	0	2	…	7
D	4,5	3,5	2,5	2,5	3,5	4,5	5,5	6,5	7,5	8,5	2	4	2	0	…	10
…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…
TITIK	17	15	13	11	9	7	5	4,5	4,5	6	14	3	7	4	…	0

ISSN:2337-7631 (printed)
ISSN: 2654-4091 (Online)

Integrasi Bobot *Mismatch* Dinamis Dalam *Edit Distance*

Jika diintegrasikan ke dalam metode-metode *approximate string matching*, metode yang diusulkan ini menbawa hasil perhitungan *edit distance* yang berbeda. Secara umum, metode *approximate string matching* (persamaan 1) dengan menggunakan bobot *mismatch* dinamis menjadi persamaan 14. Variabel $\boldsymbol{\alpha}$ akan bernilai $\boldsymbol{mw}$ jika terjadi kondisi *mismatch.*

$$D(i,j) = min\begin{cases} D(0,j), & 0 \le j \le n \\ D(i-1,j)+1 \\ D(i-1,j-1)+\alpha \\ D(i,j-1)+1 \end{cases} \quad \alpha = \begin{cases} 0, & p_i = t_j \\ mw, & lainnya \end{cases} \quad \text{..... (14)}$$

Demikian juga jika diintegrasikan dengan metode Levenstein *distance*, maka persamaan 2 menjadi persamaan 15. Nilai konstanta 1 (satu) sebagai bobot *mismatch* akan digantikan dengan nilai $\boldsymbol{mw}$.

$$Lev_{p,t}(i,j) = \begin{cases} \max(i,j) \\ \begin{cases} Lev_{p,t}(i-1,j)+1 \\ Lev_{p,t}(i,j-1)+1 \\ Lev_{p,t}(i-1,j-1)+\boldsymbol{mw}_{(p_i \ne t_j)} \end{cases} \end{cases} \text{..... (15)}$$

Dalam metode Damerau-Levenstein seperti persamaan 3, variabel $C(p_i, t_j)$ akan digantikan oleh *mw* jika ditemukan adanya *mismatch*. Persamaan 16 menunjukan perubahan persamaan 3 dengan memasukan *mw*.

$$DL(i,j) = min\begin{cases} DL(i-1,j-1)+C(p_i,t_j), jika\ i,j>0\ dan\ p_i = t_j \\ DL(i-1,j-1)+\boldsymbol{mw}, jika\ i,j>0\ dan\ p_i \ne t_j \\ DL(i,j-1)+1, jika\ i>0 \\ DL(i-1,j)+1, jika\ j>0 \\ DL(i-2,j-2)+1, if\ i,j>0\ dan\ p_i = t_{j-1} dan\ p_{i-1} = t_j \end{cases} \text{..... (16)}$$

Selanjutnya dalam metode Smith-Waterman dan Needlwman-Wunsch, konstanta *mismatch* akan digantikan oleh variabel $\boldsymbol{mw}$. Persamaan 17 dan persamaan 18 menunjukan adanya variabel mw untuk mengakomodir bobot *mismatch* dinamis.

$$SW(i,j) = \begin{cases} SW(i-1,j-1)+s(p_i,t_j), & \boldsymbol{jika}\ p_i = t_j \\ SW(i-1,j-1)+mw, & \boldsymbol{jika}\ p_i \ne t_j \\ max_{k\ge 1}\{SW(i-k,j)-W_k\}, & 1 \le i \le n, 1 \le j \le m \\ max_{l\ge 1}\{SW(i,j-l)-W_l\}, & (1 \le i \le n, 1 \le j \le m) \\ 0 \end{cases} \text{..... (17)}$$

$$NW(i,j) = \begin{cases} NW(i-1,j-1)+s(p_i,t_j), & \boldsymbol{jika}\ p_i = t_j \\ NW(i-1,j-1)+\boldsymbol{mw}, & \boldsymbol{jika}\ p_i \ne t_j \\ NW(i-1,j)-W_1 \\ NW(i,j-1)-W_1 \end{cases} \text{..... (18)}$$

4. KESIMPULAN DAN SARAN

Metode-metode *approximate string matching* mempunyai kerumitan dalam iterasi jika *string string* yang akan dibandingkan berjumlah sangat banyak misalnya dalam normalisasi NSW menjadi SW. Kekhawatiran akan adanya pembebanan komputasi karena integrasi metode yang diusulkan ini tidak perlu terjadi karena bobot *mismatch* antar-karakter pada *keyboard* akan dihitung sekali saja dan bobot *mismatch* akan disimpan dalam bentuk tabel *look-up*. Tabel *look-up* tersebut akan berbeda untuk setiap *layout keyboard*. Tabel 2 merupakan contoh tabel *look-up* untuk *layout keyboard* untuk OPPO A92. Dengan adanya pembobotan dinamis ini, masalah adanya hasil *edit distance* yang sama dalam *approximate string matching* karena adanya *mismatch* dapat terpecahkan jika yang terjadi adalah adanya kesalahan pengetikan karena keterbatasan bidang tekan pada *keyboard*.

Selanjutnya, metode ini akan diterapkan secara empiris pada beberapa metode *approximate string matching* untuk menguji kinerja empirisnya terhadap perhitungan *edit distance*. Lebih jauh akan diukur juga seberapa besar pembebanan komputasinya.

ISSN:2337-7631 (printed)
ISSN: 2654-4091 (Online)

DAFTAR PUSTAKA

[1] E. D. S. Watie, 'Komunikasi dan media sosial (communications and social media)', *Jurnal The Messenger*, vol. 3, no. 2, Art. no. 2, 2016, doi: 10.26623/themessenger.v3i2.270.

[2] M. E. Yuliana and W. Nugrahaningsih, 'PENGGUNAAN KATA TIDAK BAKU DI MEDIA SOSIAL INSTAGRAM', *INCONTECSS| ISBN: 978-623-92318-1-1*, no. 16 November, pp. 323–327, 2019.

[3] U. C. Zulkifli, 'Pengembangan Modul PreprocessingTeks untuk Kasus Formalisasi dan Pengecekan Ejaan Bahasa Indonesia pada Aplikasi Web Mining Simple Solution (WMSS)', *Jurnal Matematika, Statistika dan Komputasi*, vol. 15, no. 2, Art. no. 2, 2019, doi: 10.20956/jmsk.v15i2.5718.

[4] E. Yustika S, 'Kesalahan pengetikan (typo) seringkali dianggap sepele', 2017. https://blog.typoonline.com/kesalahan-pengetikan-typo-seringkali-dianggap-sepele-namun-bisa-berakibat-fatal/ (accessed Nov. 17, 2020).

[5] K. N. Lakonawa, S. A. Mola, and A. Fanggidae, 'NAZIEF-ADRIANI STEMMER DENGAN IMBUHAN TAK BAKU PADA NORMALISASI BAHASA PERCAKAPAN DI MEDIA SOSIAL', *J-Icon: Jurnal Komputer dan Informatika*, vol. 9, no. 1, Art. no. 1, 2021, doi: 10.35508/jicon.v9i1.3749.

[6] A. W. R. Riady, 'Normalisasi Mikroteks Berdasarkan Phonetic pada Twitter Berbahasa Indonesia Menggunakan Algoritma Jaro-Winkler Distance dan Rule Based', Skripsi, Universitas Sumatra Utara, Medan, 2019. [Online]. Available: http://repositori.usu.ac.id/handle/123456789/15418

[7] S. Priansya, 'Normalisasi Teks Media Sosial Menggunakan Word2vec, Levenshtein Distance, dan Jaro-Winkler Distance', PhD Thesis, Institut Teknologi Sepuluh Nopember, 2017. [Online]. Available: https://repository.its.ac.id/42627/

[8] K. M. M. Aung, 'Comparison of Levenshtein Distance Algorithm and Needleman-Wunsch Distance Algorithm for String Matching', PhD Thesis, University of Computer Studies, Yangon, 2019.

[9] Z. Saniyah, 'Normalisasi Mikroteks Berbentuk Singkatan pada Teks Twitter Berbahasa Indonesia Menggunakan Algoritma Longest Common Subsequences', Skripsi, Universitas Sumatra Utara, Medan, 2019. [Online]. Available: http://repositori.usu.ac.id/handle/123456789/15415

[10] R. V. Imbar, A. Adelia, M. Ayub, and A. Rehatta, 'Implementasi Cosine Similarity dan Algoritma Smith-Waterman untuk Mendeteksi Kemiripan Teks', *Jurnal Informatika*, vol. 10, no. 1, pp. 31–42, 2014.

[11] F. Marthin, 'Implemantasi Algoritma Smith Waterman Untuk Pengecekan Kesalahan Ejaan Keyword Query', Universitas Duta Wacana, Yogyakarta, 2013. [Online]. Available: https://katalog.ukdw.ac.id/4592/1/22094764_bab1_bab5_daftarpustaka.pdf

[12] P. Jokinen, J. Tarhio, and E. Ukkonen, 'A Comparison of Approximate String Matching Algorithms', *Software: Practice and Experience*, vol. 26, no. 12, Art. no. 12, 1996, doi: 10.1002/(SICI)1097-024X(199612)26:12<1439::AID-SPE71>3.0.CO;2-1.

[13] M. M. Hossain, M. F. Labib, A. S. Rifat, A. K. Das, and M. Mukta, 'Auto-correction of english to bengali transliteration system using levenshtein distance', in *2019 7th International Conference on Smart Computing & Communications (ICSCC)*, 2019, pp. 1–5. doi: 10.1109/ICSCC.2019.8843613.

[14] Q. Zhou, 'A New Approach to Sequence Local Alignment: Normalization with Concave Functions', Master's Thesis, The University of Western Ontario, 2019. [Online]. Available: https://ir.lib.uwo.ca/cgi/viewcontent.cgi?article=8541&context=etd

[15] S. B. Needleman and C. D. Wunsch, 'A general method applicable to the search for similarities in the amino acid sequence of two proteins', *Journal of molecular biology*, vol. 48, no. 3, pp. 443–453, 1970.