Deteksi Plagiarisme menggunakan Algoritma Levenshtein Distance

Yuslena Sari[1], Husnul Khatimi[2], Rizki Awlia Fajrin[3]
[123)]*Program Studi Teknologi Informasi, Fakultas Teknik, Universitas Lambung Mangkurat*
Jl. Brigjen Hasan Basry, Banjarmasin, Indonesia 70123
e-mail: yuzlena@ulm.ac.id[1)], hkhatimi@ulm.ac.id[2)], rizkyawliafajrin@gmail.com[3)]

ABSTRAK

Deteksi kesamaan dokumen untuk sistem plagiarisme termasuk dalam riset Natural Language Processing dalam bidang kecerdasan buatan. Plagiarisme banyak terjadi pada dokumen di lingkungan akademisi, begitupun yang terjadi pada PSMTS ULM. Deteksi plagiarisme diperlukan agar menjaga orisinalitas dari hasil tesis mahasiswa. Ada beberapa algoritma yang digunakan peneliti sebelumnya untuk mendeteksi plagiarisme. Namun, algoritma yang diperlukan adalah algoritma yang cepat karena yang sedang terjadi pada tesis mahasiswa relatif memiliki string yang banyak dan data tesis yang akan terus bertambah setiap saatnya mengakibatkan memperlambat kinerja algoritma. algoritma Levenshtein Distance mengungguli algoritma adaptif. Proses preprocessing yang terdiri dari metode case folding, tokenizing, stopword removal, dan stemming yang dapat melakukan estimasi proses sistem menjadi lebih cepat. Algoritma Levenshtein Distence dapat mendeteksi plagiasi dengan baik dan rata-rata lama proses sistem tanpa dilakukan preprocessing adalah 6,283 ms dan dengan preprocessing adalah 4,920 ms.

***Kata kunci**: algoritma, levenshtein distance, plagiarisme*

ABSTRACT

Document similarity detection for plagiarism systems is included in Natural Language Processing research in the field of artificial intelligence. Plagiarism occurs a lot in documents in academia, as well as what happened to the ULM PSMTS. Plagiarism detection is needed to maintain the originality of the student thesis results. There are several algorithms that previous researchers have used to detect plagiarism. However, the algorithm that is needed is fast because what is happening in the student thesis is relatively large and the thesis data will continue to increase every time, resulting in slowing down the performance of the algorithm. The Levenshtein Distance algorithm outperforms the adaptive algorithm. The preprocessing process consisting of case folding, tokenizing, stop word removal, and stemming methods can make system process estimates faster. The Levenshtein Distance algorithm can detect plagiarism well and the average system processing time without preprocessing is 6.283 ms and with preprocessing is 4.920 ms.

***Kata kunci**: algorithm, levenshtein distance, plagiarism*

PENDAHULUAN

Tindakan plagiarisme telah terjadi pada institusi sekolah dan terus meningkat, tindak plagiarisme sudah di anggap lazim oleh para siswa, data yang dikemukakan pada buku "Youth: Changing Beliefs and Behavior" bahwa telah ditemukan 58,3% siswa melakukan plagiarisme pada tugasnya dan dalam 20 tahun telah meningkat menjadi 97,5% [1]. Deteksi plagiarism dalam teks bahasa alami adalah salah satu contoh *aplikasi Natural Language Processing* (NLP) [1]–[3]. Ada beberapa algoritma yang pernah digunakan pada sistem plagiarisme, seperti algoritma Rabin Karp, Jaro -Winkler dan algoritma Levenshtein Distance [4], [5]. Permasalahan yang sedang terjadi pada tesis mahasiswa relatif memiliki string yang banyak dan data tesis yang akan terus bertambah setiap saatnya mengakibatkan memperlambat kinerja algoritma pendeteksi plagiarism. Hasil penelitian Bolle dan casey [4] menunjukkan bahwa algoritma Levenshtein Distance mengungguli algoritma adaptif. Algoritma Levenshtein Distance adalah metrik sederhana yang dapat menjadi alat aproksimasi string yang efektif. Beberapa tinjauan literature [6], [7] mengungkapkan bahwa penambahan proses preprocessing pada sistem yang menggunakan algoritma Levenshtein Distance dapat memaksimalkan kinerja sistem. Proses preprocessing yang terdiri dari metode case folding, tokenizing, stop word removal, dan stemming yang dapat melakukan estimasi proses sistem menjadi kurang dari 1 detik. Maka pada penelitian ini digunakan lah proses preprocessing dalam penerapan algoritma Levenstein Distance untuk deteksi plagiarism.

A. Natural Language Pocessing (NLP)

Pemrosesan Bahasa Alami atau NLP adalah bidang Kecerdasan Buatan yang memberi mesin kemampuan untuk membaca, memahami, dan memperoleh makna dari bahasa manusia. Deteksi kesamaan tekstual semantik adalah salah satu bidang penelitian dalam Natural Language Processing (NLP) [8], [9]. NLP mempelajari cara mengaktifkan komputer untuk memproses dan memahami bahasa yang digunakan manusia dalam kehidupan sehari-hari, memahami pengetahuan manusia, dan berkomunikasi dengan manusia dalam bahasa alami. Aplikasi

NLP termasuk pencarian informasi (IR), ekstraksi pengetahuan, sistem tanya jawab (QA), kategorisasi teks, terjemahan mesin, bantuan penulisan, identifikasi suara, komposisi, dan sebagainya [10]–[12]. Perkembangan internet dan produksi dokumen digital dalam jumlah besar telah menghasilkan kebutuhan mendesak akan pemrosesan teks cerdas, dan oleh karena itu, teori serta keterampilan NLP menjadi semakin penting.

B. Levenshtein Distance

Levenshtein Distance lebih dikenal dengan edit distance. Konsep dari Levenshtein distance yaitu mencari jumlah minimum point mutation yang diperlukan untuk merubah suatu string ke string yang lain. Point mutation tersebut adalah insertion, subtitution dan deletion [13], [14].

Levenshtein distance adalah sebuah matriks *string* yang berfungsi untuk mengukur perbedaan atau *distance* antara dua *string*. Nilai *distance* ini ditentukan oleh jumlah dari operasi-operasi perubahan yang digunakan untuk melakukan transformasi dari suatu *string* menjadi *string* lainnya. Operasi - operasi tersebut adalah penyisipan (*insertion*), penghapusan (*deletion*), atau penukaran (*subtitution*). Algoritma *Levenshtein Distance* dapat digunakan dalam mendeteksi kemiripan antara dua string yang berpotensi melakukan tindak plagiarisme [1]–[3].

Algoritma *Levenshtein Distance* bekerja dari sisi kiri atas sebuah *array*, pada dua matriks yang telah berisi *string* A dan *string* B. Berikut adalah algoritma *Levenshtein Distance* dalam mendapatkan nilai *distance* [15], [16]:

Levenshtein Distance melakukan perhitungan bobot similarity setelah mendapatkan nilai distance dari dua dokumen yang dibandingkan. Kemudian menggunakan suatu persamaan dalam menentukan bobot similarity, yaitu [7]:

$$\text{Bobot Similarity} = \left(1 - \frac{d[m,n]}{Max(S,T}\right) x100\% \qquad (1)$$

Dengan d[m,n] adalah nilai *distance*, terletak pada baris ke m dan kolom ke n, S adalah panjang *string* awal, T adalah panjang *string* target, dan Max(S,T) adalah panjang *string* terbesar antara *string* awal dan *string* target.

II. METODE PENELITIAN

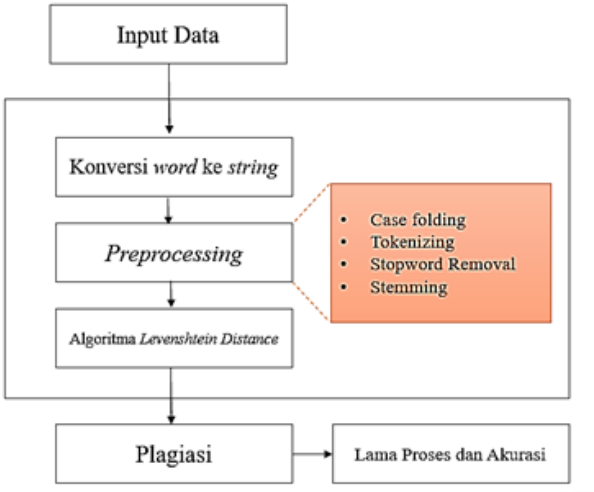
Metode yang digunakan dalam penelitian ini menggunakan algoritma *Levenshtein Distance*. Dimana algoritma *Levenshtein Distance* ini digunakan untuk mendeteksi plagiarisme tesis mahasiswa. Algoritma *Levenshtein Distance* dipilih dalam pendeteksian plagiarisme karena algoritma *Levenshtein Distance* memproses dan mempresentasikan tingkat kesamaan *string*. *Function Levenshtein Distance* sudah include pada bahasa pemrograman *Preprocessor Hypertext Protocol* (PHP). Tahap deteksi plagiarism menggunakan algoritma *Levenshtein* Dis*tance* dapat dilihat pada Gambar 1.

A. Data

Pengambilan data pada PSMT ULM, data yang diambil adalah data tesis mahasiswa yang telah mendaftar seminar secara *online*. Data tesis yang diolah adalah data yang berupa proposal tesis dan sidang akhir.

B. Covert word to string

PHP menyediakan fungsi yang mengubah *string* menjadi array. Fungsi *explode* membagi string di mana ia menemukan delimeter ditentukan. Fungsi preg_split menggunakan ekspresi reguler untuk menentukan pembatas dan menyediakan opsi untuk mengontrol *array* yang dihasilkan. Fungsi str_split membagi string menjadi elemen array dengan panjang yang sama. *String* dapat diperlakukan sebagai *array* karakter sampai batas tertentu.



Gambar 1. Tahap deteksi plagiarism

C. Preprocessing

Pada proses preprocessing adalah dimana data-data tesis yang telah di *convert* menjadi variable bertipe data string, dilakukan proses preprocessing yang terdiri dari case folding, tokenizing, stopword, stemming. Berikut dilakukan proses preprocessing sebagai berikut:

- *Case folding*

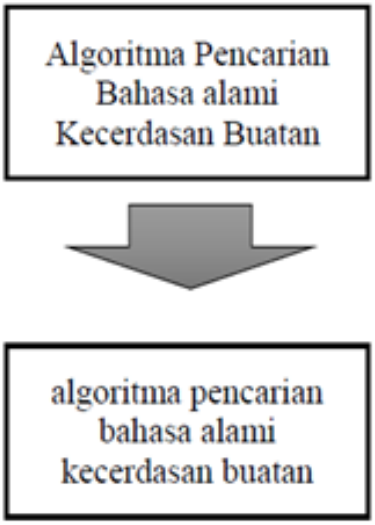
Case folding adalah proses menyetarakan kalimat menjadi huruf kecil. Berikut contoh hasil dari case folding pada Gambar 2.

- *Tokenizing*

Tokenizing adalah proses merubah string menjadi array. Dengan penggunaan array akan memudahkan sistem untuk memproses atau melakukan pengulangan pada setiap kata. Berikut ilustrasi proses tokenizing diagram tokenizing seperti pada Gambar 3.

- *Stopword removal*

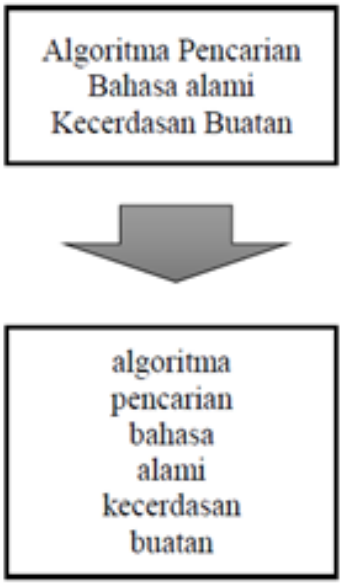
Stopword removal adalah proses menghapus kata penghubung pada sebuah dokumen. Dengan penggunaan *stopword removal* akan meningkatkan akurasi sistem dalam mendeteksi plagiarisme. List-list kata penghubung koordinatif adalah sebagai berikut: dan, serta, atau, tetapi, melainkan, padahal, sedangkan, yang, juga, adalah, dengan, pada, tersebut, dari, yaitu.



Gambar 2. Contoh ilustrasi proses *case folding*

- *Stemming*

Stemming adalah proses menghapus kata imbuhan pada sebuah kata. Dengan penggunaan stemming pada *preprocessing* dapat meningkatkan kinerja sistem dalam beroperasi.



Gambar 3. Contoh ilustrasi proses *tokenizing*

D. Proses Algoritma Levenshtein Distance

Setelah data diolah pada tahap preprocessing, maka langkah terakhir adalah data akan diproses menggunakan algoritma Levenshtein Distance, sebagai tabel 1

Seperti pada Tabel 1 adalah ilustrasi langkah untuk mencari nilai *distance*, setelah dilakukan perhitungan maka didapatkan nilai *distance* (diff), kalimat yang digunakan adalah "penggunaan uji coba algoritma pada sistem" dengan "percobaan penerapan sistem bermetode" dengan nilai 32. Maka perhitungan persentasenya adalah sebagai berikut:

i = penggunaan uji coba algoritma pada sistem
j = percobaan penerapan sistem bermetode
i = 36 (tanpa spasi)
j = 33 (tanpa spasi)
diff = 32
diff / Max(i,j) = 32/36
= 0,88888888
= 0,8889
Plagiarized value = (1 – (diff / max(i,j))) * 100
= (1 – (32 / 36)) * 100
= (1 – 0,8889) * 100
= 0,1111 * 100
= 11,11

Maka didapatlah hasil perhitungan persentase plagiasi dari kalimat "penggunaan uji coba algoritma pada sistem" dengan kalimat "percobaan penerapan sistem bermetode" adalah 11,11%. Hasil 11,11% menunjukan bahwa data yang diinputkan tidak terdeteksi plagiarisme.

III. Hasil dan pembahasan

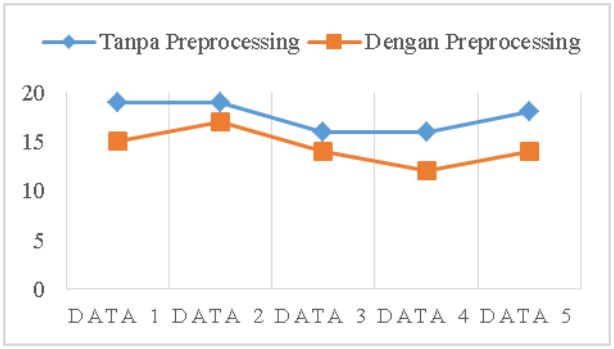
Berdasarkan dengan hasil yang ada pada pengolahan data atau hasil penelitian, dimana dari hasil tersebut akan dihitung nilai akurasinya dan lama proses sistem. Adapun untuk menghitung nilai akurasinya dan lama proses sistem maka akan dilakukan beberapa percobaan sebagai berikut.

Data 1:
Lahan rawa pasang surut (pasut) merupakan lahan yang sangat berpotensi besar mengingat luasan lahannya yang sangat besar di Indonesia
Data 2:
Kecamatan Barambai merupakan salah satu kecamatan di Kabupaten Batola yang memiliki potensi besar untuk pengembangan pertanian lahan rawa pasut.
Data 3:
Luas areal keseluruhan dari kecamatan Barambai berdasarkan hasil pengukuran pekerjaan survey pemetaan lahan pertanian pangan berkelanjutan.
Data 4:
Dengan pemanfaatan teknologi SIG diharapkan diperoleh informasi terperinci dan menyeluruh tentang kondisi rawa pasut Kecamatan Barambai.
Data 5:
Pada kecamatan Barambai digunakan pemanfaatan teknologi SIG agar dapat diperoleh informasi terperinci dan menyeluruh tentang kondisi rawa pasut

Hasil dari uji coba pendeteksian plagiarisme menggunakan algoritma *Levenshtein Distance* menggunakan 2 tahap, yaitu data yang diolah menggunakan preprocessing dan tanpa *preprocessing*.

a. Chart perubahan total kata pada *preprocessing*

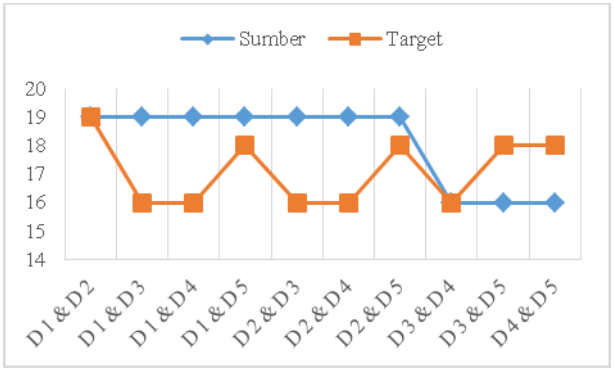
Kalimat yang telah dilakukan proses *preprocessing* akan mengalami perubahan dari struktur kalimat hingga berkurangnya jumlah kata dalam kalimat. Berikut chart total kata pada proses preprocessing seperti pada Gambar 4.



Gambar 4. Chart perubahan total kata pada *preprocessing*

b. Chart total kata source dan target tanpa preprocessing

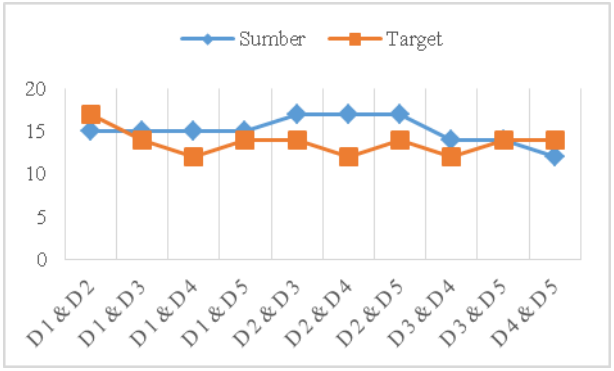
Chart untuk pembanding total sumber kata dan target tanpa dilakukan metode preprocessing pada data uji dapat dilihat pada gambar 5.



Gambar 5. Chart Total Kata Sumber & Target tanpa *Preprocessing*

c. Chart total kata source dan target dengan preprocessing

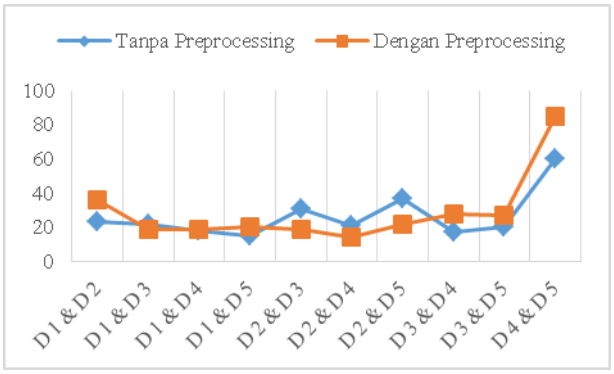
Chart untuk pembanding total kata pada source dan target dengan dilakukan metode *preprocessing* pada data uji terlihat pada gambar 6.



Gambar 6. Chart Total Kata Source & Target dengan *Preprocessing*

d. Chart nilai plagiasi

Chart untuk pembanding nilai plagiasi pada data uji dengan menggunakan preprocessing dan tanpa preprocessing terlihat pada gambar 7.

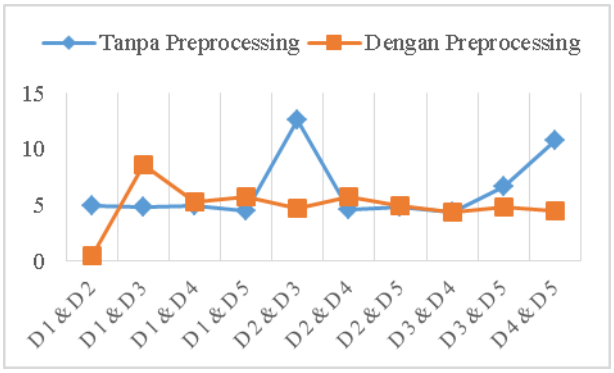


Gambar 7. Chart Nilai Plagiasi

Seperti pada Gambar 7, hasil dari uji coba dapat diambil kesimpulan rata-rata nilai plagiasi tanpa melakukan metode preprocessing adalah 26,4% dan rata-rata nilai plagiasi dengan melakukan metode preprocessing adalah 28,9% yang artinya hasil uji coba plagiasi rata-rata dibawah 50%, hasil ini menunjukan data uji coba tidak terdeteksi sebagai plagiasi. Akan tetapi pada uji coba Data 4 & Data 5, nilai plagiasi tanpa preprocessing menunjukan 60% dan dengan tahapan preprocessing adalah 85%.

e. Chart lama proses sistem

Chart untuk pembanding lama proses sistem saat sedang menguji data plagiasi dapat dilihat pada gambar 8.



Gambar 8. Chart Lama Proses

Seperti pada Gambar 8, hasil dari uji coba lama proses sistem saat bekerja dapat diambil kesimpulan rata-rata lama proses sistem tanpa dilakukan *preprocessing* adalah 6,283 ms dan dengan *preprocesisng* adalah 4,920 ms. Maka dalam hal ini penggunaan metode preprocessing lebih efektif untuk meningkatkan kinerja proses sistem.

IV. KESIMPULAN

Berdasarkan hasil penelitian dan diskusi, dapat disimpulkan deteksi plagiarism menggunakan algoritma *Lavenshtein Distance* ini menggunakan *preprocessing* proses yaitu *case folding*, *tokenizing*, *stopword removal* dan *stemming* lebih efektif. Hasil akurasi dengan tahap *preprocessing* bernilai lebih tinggi dibanding tanpa *preprocessing*. Penggunaan *preprocessing* dapat meningkatkan kinerja proses sistem dengan rata-rata waktu 4,920 ms. Penggunaan tahapan *preprocessing* dapat menstabilkan proses deteksi plagiasi dibandingkan dengan tanpa *preprocessing*.

DAFTAR PUSTAKA

[1] X. Yao, M. H. Yap, and Y. Zhang, "An Empirical Study to Evaluate Structural Similarity for Source Code Translation," *TIMES-iCON 2019 - 2019 4th Technol. Innov. Manag. Eng. Sci. Int. Conf.*, pp. 1–5, 2019, doi: 10.1109/TIMES-iCON47539.2019.9024512.

[2] A. Altheneyan and M. E. B. Menai, *Evaluation of State-of-the-Art Paraphrase Identification and Its Application to Automatic Plagiarism Detection*, vol. 34, no. 4. 2020.

[3] P. Sunilkumar and A. P. Shaji, "A Survey on Semantic Similarity," *2019 6th IEEE Int. Conf. Adv. Comput. Commun. Control. ICAC3 2019*, 2019, doi: 10.1109/ICAC347590.2019.9036843.

[4] T. Bollé and E. Casey, "Using computed similarity of distinctive digital traces to evaluate non-obvious links and repetitions in cyber-investigations," *DFRWS 2018 EU - Proc. 5th Annu. DFRWS Eur.*, vol. 24, pp. S2–S9, 2018, doi: 10.1016/j.diin.2018.01.002.

[5] T. Tinaliah and T. Elizabeth, "Perbandingan Hasil Deteksi Plagiarisme Dokumen dengan Metode Jaro-Winkler Distance dan Metode Latent Semantic Analysis," *J. Teknol. dan Sist. Komput.*, vol. 6, no. 1, pp. 7–12, 2018, doi: 10.14710/jtsiskom.6.1.2018.7-12.

[6] A. B. Nasser, A. Alsewari, and K. Z. Zamli, *Computing, Analytics and Networks*, vol. 805. Springer Singapore, 2018.

[7] R. Haldar and D. Mukhopadhyay, “Levenshtein Distance Technique in Dictionary Lookup Methods: An Improved Approach,” no. January 2011, 2011, [Online]. Available: http://arxiv.org/abs/1101.1232.
[8] M. A. Arabacı, E. Esen, and M. S. Atar, “Kelime Gömevi Yöntemi Kullanarak Benzer Cümle Tespiti Detecting Similar Sentences Using Word Embedding,” *2018 26th Signal Process. Commun. Appl. Conf.*, pp. 1–4.
[9] G. Majumder, P. Pakray, A. Gelbukh, D. Pinto, and D. Puebla, “Semantic Textual Similarity Methods , Tools , and Applications :,” vol. 20, no. 4, pp. 647–665, 2016, doi: 10.13053/CyS-20-4-2506.
[10] M. C. Lee, J. W. Chang, and T. C. Hsieh, “A Grammar-Based Semantic Similarity Algorithm for Natural Language Sentences,” vol. 2014, 2014.
[11] S. Kim, I. Park, and B. Y. Id, “SAO2Vec : Development of an algorithm for embedding the subject – action – object (SAO) structure using Doc2Vec,” pp. 1–26, 2020, doi: 10.1371/journal.pone.0227930.
[12] M. Antonio, C. Soares, and F. S. Parreiras, “A literature review on question answering techniques , paradigms and systems,” *J. King Saud Univ. - Comput. Inf. Sci.*, 2018, doi: 10.1016/j.jksuci.2018.08.005.
[13] H. A. Maarif, R. Akmeliawati, Z. Z. Htike, and T. S. Gunawan, “Complexity algorithm analysis for edit distance,” *Proc. - 5th Int. Conf. Comput. Commun. Eng. Emerg. Technol. via Comp-Unication Converg. ICCCE 2014*, pp. 135–137, 2015, doi: 10.1109/ICCCE.2014.48.
[14] S. Jakšić, E. Bartocci, R. Grosu, T. Nguyen, and D. Ničković, “Quantitative monitoring of STL with edit distance,” *Form. Methods Syst. Des.*, vol. 53, no. 1, pp. 83–112, 2018, doi: 10.1007/s10703-018-0319-x.
[15] S. Ontañón, *An overview of distance and similarity functions for structured data*, no. 0123456789. Springer Netherlands, 2020.
[16] M. A. Syaekhoni, C. Lee, and Y. S. Kwon, “Analyzing customer behavior from shopping path data using operation edit distance,” *Appl. Intell.*, vol. 48, no. 8, pp. 1912–1932, 2018, doi: 10.1007/s10489-016-0839-2.