Journal of Information Systems Engineering and Management
2025, 10(28s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

An Enhancement of the Jaro-Winkler Fuzzy Searching Algorithm Applied in Library Search Engine

Karl Benedict K. Malaga [1], Korinne L. Verdillo [2], Elsa S. Pascual [3]

[1] *Student in Computer Science, College of Information Systems and Technology Management, University of the City of Manila, Manila, Philippines. Email: malaga.karlbenedict@gmail.com, Orcid Id: 0009-0008-8702-9389*

[2] *Student in Computer Science, College of Information Systems and Technology Management, University of the City of Manila, Manila, Philippines. Email: korinnverdillo@gmail.com, Orcid Id: 0009-0008-6093-6501*

[3] *Faculty in Computer Science, College of Information Systems and Technology Management, University of the City of Manila, Manila, Philippines. Email: espascual@plm.edu.ph*

**Corresponding Author: Karl Benedict K. Malaga*

ARTICLE INFO

Received: 30 Dec 2024

Revised: 12 Feb 2025

Accepted: 26 Feb 2025

ABSTRACT

The Jaro-Winkler algorithm is widely used for approximate string matching, offering reliable similarity calculations between two strings. However, its performance declines with increasing string length due to bias against longer strings and its reliance on prefix similarity, which neglects significant suffix matches. This paper presents an Enhanced Jaro-Winkler algorithm that addresses these challenges by integrating a Rabin-Karp Rolling Hash – inspired technique and applying suffix weights to balance the prefix bias. Experimental evaluations using 100 words commonly found in book titles demonstrate the enhanced algorithm's robustness across varying fuzzy match thresholds (0.7, 0.8, and 0.9). Unlike the traditional algorithm, where higher thresholds reduce match accuracy, the enhanced algorithm consistently achieves 100% accuracy in identifying titles regardless of query position or threshold. Additionally, it showcases superior performance by improving the quality and quantity of retrieved results by a significant number of titles compared to the traditional approach. These advancements highlight the algorithm's potential for improving search performance in applications requiring precise and flexible string matching.

Keywords: Approximate String Matching, Fuzzy Matching, Fuzzy Logic, Jaro-Winkler.

INTRODUCTION

Fuzzy Searching or Fuzzy Matching is another term for Approximate String Matching. It is a technique used to compare strings that partially match rather than exactly. The algorithms in Fuzzy Matching aim to determine the degree of closeness between two strings and decide whether they are considered as fuzzy match [1][17]. This matching method is suitable for searching database items that may have spelling mistakes, typographical differences, or other errors caused by humans or computers.

The Jaro-Winkler algorithm is an approximate string-matching algorithm that calculates how two strings are similar. It calculates a percentage of similarity based on the string's length and number of matching and unmatching characters, 80% being the default threshold to be considered as a fuzzy match [18]. The Jaro distance was introduced by Matthew A. Jaro as a record-linkage methodology for census to match large number of records quickly and accurately [2]. It was later enhanced by William E. Winkler, by developing a Winkler scale which increases the similarity score when the two strings have common prefixes [3].

Despite its wide range of applications, the Jaro-Winkler algorithm still faces multiple challenges affecting its performance. While it demonstrates good performance with short strings, its efficiency declines as the length of the string being compared increases [4][5]. Additionally, comparing a short and single string to a set of multiple strings results in lower similarity score even though the strings are closely related to each other. This behavior occurs because the Jaro-Winkler algorithm evaluates string similarity based on character order and proximity [6]. A study by

Karakasidis and Pitoura [7] highlights the importance of recognizing potential bias in string comparison methods. The existence of potential bias can affect the results generated by an algorithm hence, it is crucial to also address this issue. Other than these challenges, the Jaro-Winkler remains a highly effective algorithm, consistently delivering strong performance. Furthermore, these existing challenges can be seen as opportunities for further enhancement of the algorithm and expand its capabilities.

The Jaro-Winkler algorithm demonstrates potential for broader applications beyond its current use. This study focuses on exploring how the algorithm might be adapted for library search engines, given the importance of efficient search methods in modern libraries. By leveraging the algorithm, library search engines can deliver enhanced services to meet the needs of contemporary users.

RELATED WORKS

Ali et.al. [10] applied the Jaro-Winkler fuzzy matching in correlating and integrating a database. The Jaro-Winkler was used to calculate the similarity in two different data, where it turns one if the comparison indicates match or zero if there is no similarity. The study concluded that Jaro-Winkler is considered as the best algorithm for fuzzy matching, however its speed is dependent on the length of the strings being compared [8]. Manaf et al. [6] compared the effectiveness of the Jaro-Winkler and Rabin-Karp algorithms for detecting document similarity. The Jaro-Winkler algorithm measures similarity between two strings by calculating their length, identifying character matches, accounting for transpositions (Jaro), and applying a prefix scale adjustment (Winkler). In contrast, the Rabin-Karp algorithm uses the rolling hash technique, which calculates the hash of a specific pattern and checks if that hash exists in the target string. Their comparison showed that Jaro-Winkler is not well-suited for long or non-sequential patterns, but excels with shorter patterns, such as names. Additionally, they noted that Jaro-Winkler performs significantly faster than Rabin-Karp. Using these two algorithms, Leonardo and Hansun [9] compared the effectiveness of detecting plagiarism in text documents. Their experiment involved analyzing text, docx, and pdf files, with sizes ranging from under 1000 KB to over 1000 KB. The results consistently showed Rabin-Karp outperforming Jaro-Winkler in terms of both processing time and average similarity score. Based on these findings, the researchers concluded that Rabin-Karp is a more effective algorithm for document plagiarism detection. This supports Agbehadji et al.'s [11] assertion that Jaro-Winkler is better suited for comparing short strings, such as names or individual words, but its accuracy declines when working with larger datasets.

Rozinek and Mares [12] addressed the problem of Jaro and Jaro-Winkler where it overlooks the sequence of characters in the matching window when comparing two strings. In their study, Convolutional Jaro (ConvJ) and Convolutional Jaro-Winkler (ConvJW) were introduced to address the issue where character sequence is affecting its accuracy in string matching. The enhanced version of both algorithms utilizes Gaussian Weighting for calculating the positional proximity of each matching character. Results demonstrate improvement in computational efficiency as well as its accuracy compared to the conventional Jaro and Jaro-Winkler. The Convolutional Jaro performed 7x faster than the conventional Jaro and had a 10% increase in F1-score. Both ConvJ and ConvJW displayed exceptional performance in a wide range of datasets.

To address the issue of srting matching for names with multiple variations and errors in spelling, Christen [13] conducted an experimental comparison on some of the existing name matching techniques. Various Phonetic Encoding and Pattern Matching techniques including Soundex, Levenshtein Distance, and Jaro-Winkler were evaluated based on matching accuracy and computational performance. The study had mixed results which revealed that no particular matching technique is considered the best. The characteristics of names, and the computational requirement must be considered when selecting a technique. Although it is recommended to use Jaro-Winkler or q-grams if the algorithm's execution time is a priority.

Friendly (2019) utilized result indexing to save the Jaro-Winkler results for all queries that the users enter. Their enhancement decreased the search time 90-92% resulting in much faster access. The access time for querying a word for the first time is the same as a normal search, but when the results have already been saved, the access speed reduces greatly. [16]

A study conducted by Yancey (2005) evaluated string comparators to check which has the best performance. The study used different string comparators including variations of Jaro-Winkler, Edit Distance, and Hybrid Comparators. A test deck that was clerically matched was used to determine the best comparator. Each algorithm

was applied and assessed to check if it would also identify the clerically matched records as matches. The study highlighted that the hybrid comparator performs slightly better among all the variations, however the downside is that it takes longer to run. [19]

METHODOLOGY

This study used quantitative research design with a descriptive-comparative approach. The dataset of book titles was used in the experimental setup to test the base Jaro and Jaro-Winkler algorithms along with the Levenshtein Distance, Soundex algorithm and the proposed enhanced Jaro-Winkler algorithm. Levenshtein and Soundex were selected as they are the currently available fuzzy matching functions in SQL [15]. Despite the comprehensive approach, the study also has its limitations. The study only prioritized accuracy, and not speed as, in comparison, the enhanced algorithm has more processes than the base algorithm.

The metric used to measure and compare the algorithms are the number of matches the algorithms returned that included the exact search query over the total number of titles that include the search query.

$$accuracy = \frac{results\ that\ include\ the\ exact\ query}{total\ number\ of\ titles\ with\ the\ exact\ query}$$

If the exact match is found in the title, it is guaranteed that the fuzzy matches were considered during the process. Further analysis was conducted after the experimental process to determine if the enhanced algorithm was, in fact, better than the base algorithms. Moreover, despite speed being a limitation of this study, the researchers still measured the processing time to compare against the existing algorithms.

The researchers used Python as the main programming language for the enhancement of the Jaro-Winkler algorithm, utilizing libraries such as the Levenshtein library (including base Jaro and Jaro-Winkler algorithms) and the Soundex library from pypi.org.

3.1) Base Jaro-Winkler Algorithm:

The Jaro-Winkler Algorithm consists of two techniques, the Jaro Distance and the Winkler Scale. The Jaro Distance calculates the fuzzy similarity of two strings based on their matching characters and transpositions. After calculating the Jaro Distance, the Winkler Scale is added to increase the similarity score if the two strings have similar prefixes (maximum of 4 characters).

$$jd = \left(\frac{m}{length(\text{target})} + \frac{m}{length(\text{referent})} + \frac{m-t}{m}\right) \cdot \frac{1}{3}$$

Where m is the number of matching characters and t is the number of transpositions. To consider matching characters, a maximum distance window is used. This distance looks symmetrically into the characters before and after the current index to determine if there are matching characters within the window. The maximum distance formula is as follows:

$$maxDistance = \left\lfloor \frac{\max(target, referent)}{2} \right\rfloor - 1$$

The Winkler Scale, on the other hand, adds more emphasis and weight on words that have the same prefix with a maximum of 4 letters. Using the formula:

$$jw = jd(target, referent) + L \times P \times (1 - jd(target, referent))$$

Where L is the length of the common prefix at the start of the string up to a maximum of 4 characters and P is the scaling factor which is usually 0.1 by default.

3.1.1) Pseudocode of Base Jaro-Winkler Algorithm

```
Base Jaro Winkler Algorithm
jaro_winkler(target, referent):
        tLength = get the length of target
        rLength = get the length of referent
        maxD = get the max range that will be considered as "matched characters"
                floor(max(tLength, rLength) / 2) – 1
        m = get matched characters with per s1 character ± maxD
        t = get number of transposable characters
        Perform the Jaro Distance on the two strings:
                jd = (m/s1 + m/s2 + (m−t)/m) × 1/3
        Perform the Winkler Scaler:
                prefix = 0
            for I in range 4:
                        if target[i] == referent[i]:
                                prefix += 1
        jw = jd + prefix * 0.1 * (1-jd)

        return jw;
```

3.1.2) Flowchart of the Base Jaro-Winkler Algorithm

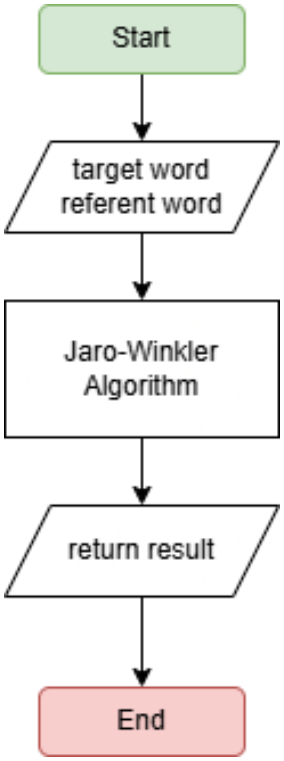

Figure 1: Base Jaro-Winkler Algorithm Flowchart

Figure 1 shows the flow chart of the existing process of the Jaro Winkler Algorithm. The target and referent undergo data preparation and gets passed to the algorithm and a result will be returned.

3.2) Proposed Enhanced Jaro-Winkler Algorithm:

The proposed enhanced Jaro-Winkler Algorithm consists of two techniques:

1.) The algorithm performs a rolling comparison (inspired by the Rabin Karp algorithm) wherein it compares the target to a substring of the referent, depending on the number of words the target has.

Table 1: Rolling Comparison Example with 1-word target

Iteration	Target	Referent	Score	Match?
1	“coloring”	“A”	0%	No
2	“coloring”	“simple”	43.1%	No
3	“coloring”	“coloring”	100%	Yes
4	“coloring”	“book”	58.3%	No

Example 1: Target = “coloring” | Referent = “A simple coloring book” | Threshold = 0.8

Table 2: Rolling Comparison Example with 2-word target

Iteration	Target	Referent	Score	Match?
1	“colour bok”	“A simple”	40.8%	No
2	“colour bok”	“simple coloring”	47.8%	No
3	“colour bok”	“coloring book”	91.8%	Yes

Example 2: Target = “colour bok” | Referent = “A simple coloring book” | Threshold = 0.8

2.) The suffix scale enhancement is only applicable if the prefix scale does not return a value greater than 0.8. The algorithm checks three criteria before applying the suffix scale [4]:

1. Both target and referent lengths are greater than 5 characters
2. There are at least 2 matching characters other than the accepted prefix
3. The matching characters must be greater than or equal to the length of the shorter string excluding the accepted prefix

The algorithm uses the same formula it used for the prefix scale but for the last characters (maximum 4) to get the modified and enhanced score.

3.2.1) Pseudocode of Proposed Enhanced Jaro-Winkler Algorithm

```
suffix_weight(target, referent, matches, prefix, jd):
        if (
                lengths of target and referent > 5,
                matches – prefix ≥ 2,
                matches – prefix ≥ (short-prefix)/2
        ):
                suffix = number of similar in last 4 letters
                jw = jd + suffix * 0.1 * (1-jd)
                return jw
```

```
jaro_winkler(target, referent):

        tSplit = words from target separated with spaces
        rSplit = words from referent separated with spaces
        tLength = length of target
        rLength = length of referent
        tsLength = length of tSplit
        rsLength = length of rSplit
        maxJW = 0.0

        if tsLength < rsLength:
                for word in rSplit:
                        group = word + [(len of tSplit – 1) words from rSplit]
                        gLength = length of group
                        perform Jaro-Winkler for each group
                        if jw > maxJW:
                                maxJW = jw
                        jw = suffix_weight(target, referent, m, prefix, jd)
                        if jw > maxJW:
                                maxJW = jw

                return maxJW

        else:
                Perform base Jaro-Winkler

                if jw > maxJW:
                        maxJW = jw
                jw = suffix_weight(target, referent, m, prefix, jd)
                if jw > maxJW:
                        maxJW = jw

                return maxJW
```

3.2.2) Flowchart of the Proposed Enhanced Jaro-Winkler Algorithm

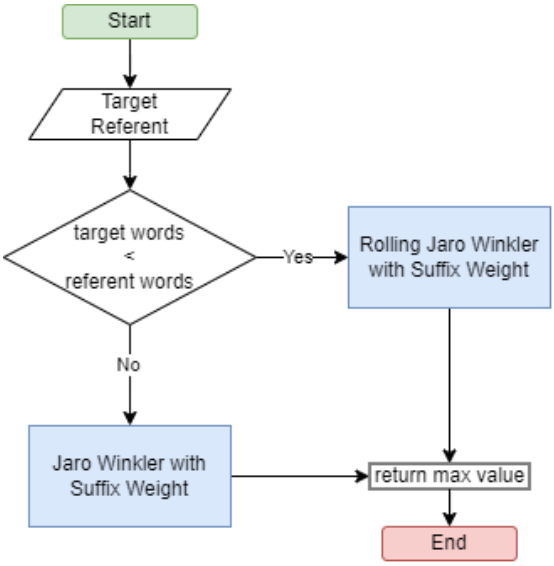

Figure 2: Proposed Enhanced Jaro-Winkler Flowchart

Figure 2 shows the proposed enhancement with two techniques, the Rolling Jaro Winkler and the Suffix Weight. If the target words are less than the referent words, the algorithm will execute the Rolling Jaro Winkler with additional Suffix Weighting, else, it will execute the plain Jaro Winkler but with additional Suffix Weighting. After the additional processes, the max value of the results will be returned.

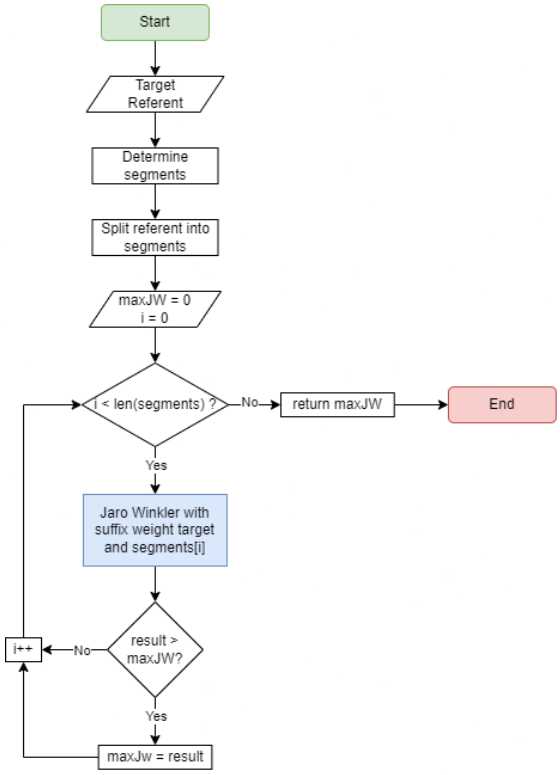

Figure 3: Rolling Jaro Winkler with Suffix Weight Enhancement Flowchart

Figure 3 shows the algorithm for the Rolling Jaro Winkler. It iterates the referent according to the number of words in the target. After all iteration, the algorithm will return the highest match value.

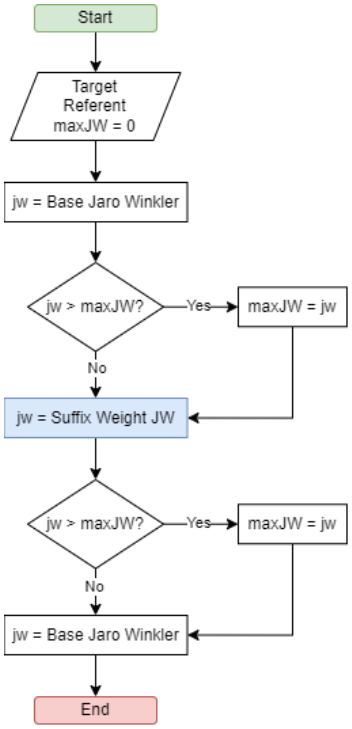

Figure 4: Jaro Winkler with Suffix Weight Enhancement Flowchart

Figure 4 shows the flow chart for the Jaro Winkler with Suffix Weight enhancement. It first processes the prefix weight, then compare if the suffix weight gets a better score. The larger score will be the one returned.

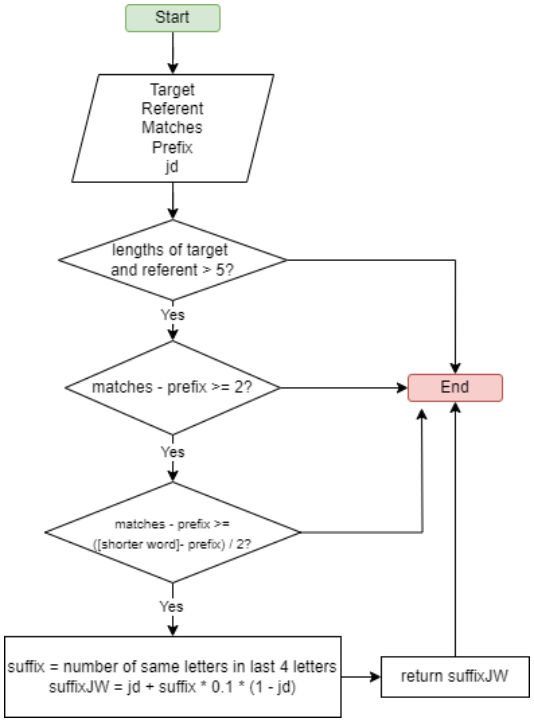

Figure 5: Suffix Weight Enhancement Flowchart

Figure 5 shows the process of the Suffix Weight enhancement. It first checks the validity of the two strings then proceeds to adding the weight. The formula that is used for the suffix score is same to the one used in the prefix score

The enhanced version of the Jaro-Winkler algorithm is composed of several optimizations to further improve its accuracy for fuzzy matching. In the traditional Jaro-Winkler, when comparing a shorter string to a longer string, each string is treated as a whole, regardless of whether it has substrings. Consequently, for cases where the exact match of the target string is located in the middle or the end of the referent string, matches often go unidentified and excluded from search results, as they are given a lower similarity score. The aim of the study was to optimize the algorithm's searching capabilities. Instead of treating the strings as a whole, the enhanced version divides long strings into substrings before performing a comparison, as shown in 3.2 Proposed Enhanced Jaro-Winkler Algorithm. This method allows the algorithm to take account for finding the match of the shorter string, that may be located in the middle or towards the end of the longer string.

The enhanced version of the algorithm was tested in more than 100k book titles obtained from Kaggle, varying from two-word titles to twenty-word titles. The experiment conducted to test the performance of the enhanced Jaro-Winkler was somewhat similar to a study conducted by the Yancey (2005) where a set of records are used to check whether a string comparator can accurately match strings that are considered clerically matched. The top 100 most frequently occurring words in book titles were selected through an automated program, the test involves counting the exact matches of the words in the database. The underlying theory is that the optimized Jaro-Winkler comparator should identify the same number of matches as those found by the automated program. This aims to demonstrate that the enhanced comparator can successfully match shorter strings even when their corresponding matches are located at the middle or the end of longer strings.

Table 3: Top 100 words with 4 or more letters in book titles

Book	What	People	Secrets	Work
Guide	Little	Good	Secret	Reading
Your	Books	House	Make	Every

Novel	Best	Classics	Tales	Over
From	America	Handbook	When	Faith
Life	Cookbook	Power	Garden	Night
Series	That	Business	Level	Most
With	First	Children	Food	Gods
American	Women	Health	Mysteries	Making
World	Family	Easy	Journey	Child
Edition	Other	Better	Golden	Healthy
Love	Time	Collection	True	Year
History	Christmas	Kids	Dictionary	School
Stories	More	Science	Gardens	Things
Home	Living	Know	Everything	Washington
Story	Cooking	Americas	Illustrated	Young
Recipes	Library	Country	Volume	Read
Great	Mystery	Guides	Readers	Century
Complete	Heart	Years	Last	Modern
About	Bible	Through	Into	Adventures

Table 3 shows the top 100 most frequently occurring words, with 4 or more letters found in book titles. These common words are used as the target string when utilizing the optimized version of the algorithm. The list of titles above is obtained from an automated frequency counting using a program. The exact matches of these words were counted and is compared to the returned matches of the enhanced JW.

The dataset for testing is extracted from Kaggle.com specifically a dataset entitled “Books Dataset” where information was scraped from wonderbk.com a popular online bookstore. This dataset contains 103,063 records, with key attributes such as title, authors, description, category, publisher, starting price, and publish date. [14]

RESULTS

To determine the performance of the algorithm, the data was collected from the experiment conducted as described in section 3.2.1. Below is a table containing the average metrics results of searching through a set of data containing more than 100k book titles as the target strings, and the top 100 most frequently occurring words in titles as referent string. The enhanced Jaro-Winkler algorithm is compared to Base Jaro, Base Jaro-Winkler, Levenshtein Distance, and Soundex. The Base Jaro and Base Jaro-Winkler are divided into three versions, differing only in their threshold values. The tests conducted measured the average search results, returned exact matches, actual exact matches, exact match accuracy, and time of execution.

Table 4: Average statistics result from testing 100 words against the dataset

	Search Results	**Returned Exact Matches**	**Actual Exact Matches**	**Exact Match Accuracy**	**Time of Execution**
Base JW (0.7)	688.39	113.65	890.77	14.48%	1.294s
Base JW (0.8)	137.01	80.82	890.77	11.14%	
Base JW (0.9)	3.61	1.95	890.77	0.3%	

	Search Results	Returned Exact Matches	Actual Exact Matches	Exact Match Accuracy	Time of Execution
Base Jaro (0.7)	276.91	79.58	890.77	10.36%	0.178s
Base Jaro (0.8)	9.66	4.27	890.77	0.7%	
Base Jaro (0.9)	0.64	0.32	890.77	0.06%	
Levenshtein	44.35	0.43	890.77	0.07%	0.184s
Soundex	4.39	1.00	890.77	0.02%	2.357s
Enhanced JW (0.7)	18,344.97	890.77	890.77	100%	5.179s
Enhanced JW (0.8)	3,379.3	890.77	890.77	100%	
Enhanced JW (0.9)	1,522.58	890.77	890.77	100%	

Average Search Results

The values collected for search results are the average number of book titles that are returned by each algorithm. The enhanced Jaro-Winkler with a 0.7 threshold shows the highest average of search results among the other algorithms, followed by the Base Jaro-Winkler with a 0.7 threshold with an average of 688.39 returned search results. The algorithm with the lowest returned search results is the Base Jaro with a 0.9 threshold, which only returns 0.64 on average. The three enhanced versions with three different thresholds show a significant increase in returned search results.

Average Returned Exact Matches

The values in the "Returned Exact Matches" column represent the average number of returned search results that are identified by each algorithm as a 100% exact match with the target string. The three enhanced versions show the same highest average number of returned exact matches while the Levenshtein and Base Jaro (0.9) have the lowest. Both Base Jaro and Base Jaro-Winkler with a 0.7 threshold return a higher average of returned exact matches compared to those with 0.8 and 0.9 thresholds.

Average Actual Matches

The "Actual Matches" values represent the number of correct matches that are considered as the actual matches for the target strings. These indicate the number of valid matches for each of the 100 book titles. All algorithms share the same "Actual Matches" values since they are evaluated using a single, consistent dataset for testing.

Exact Match Accuracy

The values in the "Exact Match Accuracy" column represent the percentage of correct matches accurately and correctly returned by each algorithm. Compared to the traditional version, the enhanced Jaro-Winkler demonstrates higher matching accuracy, as it successfully identifies all exact matches in the database. All three versions with different thresholds achieve 100% match accuracy, consistently returning all valid matches. Base Jaro (0.7) with 14.48% exact matching accuracy, is the second highest, although significantly lower than the enhanced Jaro-Winkler. The Levenshtein, Base Jaro (0.9), and Soundex algorithms have the lowest matching accuracy, with only 0.07%, 0.06% and 0.02%, respectively.

Time of Execution

The “Time of Execution” values represent the average time, in seconds, that each algorithm took to perform the fuzzy search. All three enhanced versions show a significant increase in execution time, with an average of 5.179 seconds. The Base Jaro (0.7, 0.8, and 0.9) performs the fastest execution among all the algorithms with an average of 0.178s, followed by Levenshtein with an average of 0.184s. Soundex is the second slowest in execution time, with an average of 2.357 seconds. In the middle, is the Base Jaro-Winkler (0.7, 0.8, and 0.9), with an average of 1.294s.

CONCLUSION

This paper presented an enhancement of the Jaro-Winkler algorithm using two methods, the Rolling Jaro Winkler Technique and Suffix Weighting. This approach has been assessed by using a dataset with 100k data comparing the speed and accuracy of several other algorithms including the Levenshtein Distance, Soundex, Jaro Distance, and Base Jaro Winkler (0.7, 0.8, and 0.9 thresholds in Jaro and Jaro Winkler included). The results indicate that despite having an increase in execution time, the Enhanced Jaro Winkler showcased a consistent accuracy of 100%, finding keywords regardless of the position in the referent word. Finding the exact match guarantees finding the fuzzy match in the referent string.

ACKNOWLEDGEMENT

We would like to express our gratitude to Karl Benedict Malaga and Korinne Verdillo for their hard work and dedication to this work. Our appreciation goes to our thesis coordinator, Vivien Agustin, for their guidance and support throughout this process, as well as to our defense panelist, Raymund Dioses, for their valuable insights and recommendations, which helped refine and improve our work. Finally, we are also deeply thankful to our advisor, Elsa Pascual, for their insightful feedback and constructive criticism, which greatly enhanced the quality of this paper.

REFERENCES

[1] Pikies, M., & Ali, J. (2020). Analysis and safety engineering of fuzzy string-matching algorithms. ISA Transactions, 113, 1–8. https://doi.org/10.1016/j.isatra.2020.10.014

[2] Jaro, M. A. (1987). Advances in Record-Linkage methodology as applied to matching the 1985 census of Tampa, Florida. Journal of the American Statistical Association, 84(406), 414–420. https://doi.org/10.1080/01621459.1989.10478785

[3] Winkler, William. (1994). Advanced Methods for Record Linkage. https://www.researchgate.net/publication/245534659_Advanced_Methods_For_Record_Linkage

[4] Ranzijn, B.A. (2013, October 30). A Geocoding Algorithm Based on A Comparative Study Of Address Matching Techniques. Econometrie. Retrieved from http://hdl.handle.net/2105/14891

[5] Yulianto, Muhamad & Nurhasanah, Nurhasanah. (2021). The Hybrid of Jaro-Winkler and Rabin-Karp Algorithm in Detecting Indonesian Text Similarity. Jurnal Online Informatika. 6. 88. https://doi.org/10.15575/join.v6i1.640

[6] Leonardo, B., & Hansun, S. (2017, February). Text documents plagiarism detection using Rabin-Karp and Jaro-Winkler Distance Algorithms. ResearchGate. https://www.researchgate.net/publication/316681173_Text_Documents_Plagiarism_Detection_using_Rabin-Karp_and_Jaro-Winkler_Distance_Algorithms

[7] Novyantika, R. D., & Isa, S. M. (2023, January). Improve Data Text Quality by Applying Text Pre- Processing Method (Case Study). International Journal of Engineering Trends and Technology. https://ijettjournal.org/Volume-71/Issue-1/IJETT-V71I1P209.pdf

[8] Manaf, K., Pitara, S., Subaeki, B., Gunawan, R., Rodiah, & Bakhtiar. (2019). Comparison of Carp Rabin algorithm and Jaro-Winkler distance to determine the equality of Sunda languages. 2019 IEEE 13th International Conference on Telecommunication Systems, Services, and Applications (TSSA), 77–81. https://doi.org/10.1109/TSSA48701.2019.8985470

[9] Karakasidis, A., & Pitoura, E. (2019). Identifying Bias in Name Matching Tasks. Open Proceedings. http://dit.unitn.it/~pavel/OM/articles/EDBT19_paper_213.pdf#page=2.42

[10] Ali, M., Saikia, A., Baruah, R., & Sarma, U. (2018, August). Jaro Winkler Fuzzy match algorithm to calculate a similarity index between two strings using open-source platform. International Journal for Innovative Research In Multidisciplinary Field. https://www.ijirmf.com/wp-content/uploads/IJIRMF201808019.pdf

[11] Agbehadji, I. E., Yang, H., Fong, S., & Millham, R. (2018). The Comparative Analysis of Smith-Waterman Algorithm with Jaro-Winkler Algorithm for the Detection of Duplicate Health Related Records. ResearchGate. https://www.researchgate.net/publication/327712322_The_Comparative_Analysis_of_Smith-Waterman_Algorithm_with_Jaro-Winkler_Algorithm_for_the_Detection_of_Duplicate_Health_Related_Records

[12] Rozinek, O., & Mares, J. (2024, May). Fast and precise convolutional Jaro and Jaro-Winkler similarity. ResearchGate. https://www.researchgate.net/publication/380360789_Fast_and_Precise_Convolutional_Jaro_and_Jaro-Winkler_Similarity

[13] Christen, P. (2006). A comparison of personal name matching: Techniques and practical issues. ResearchGate. https://www.researchgate.net/publication/215992032_A_Comparison_of_Personal_Name_Matching_Techniques_and_Practical_Issues
[14] Books Dataset. (2023, December 20). Kaggle. https://www.kaggle.com/datasets/elvinrustam/books-dataset
[15] Wei, L. W. (2024, April 29). Mastering Fuzzy Match Techniques in SQL - Explore the intricacies of implementing fuzzy match techniques in SQL, enhancing data matching accuracy for better data management and analysis. - SQLPad.io. SQLPad. https://sqlpad.io/tutorial/mastering-fuzzy-match-techniques-in-sql/
[16] Friendly, F. (2019). Jaro–Winkler Distance Improvement For Approximate String Search Using Indexing Data For Multiuser Application. Radware bot manager Captcha. https://iopscience.iop.org/article/10.1088/1742-6596/1361/1/012080/meta
[17] Hall, P. a. V., & Dowling, G. R. (1980). Approximate string matching. ACM Computing Surveys, 12(4), 381–402. https://doi.org/10.1145/356827.356830
[18] Fuzzy matching. (n.d.). https://www.soluling.com/Help/FuzzyMatch.htm
[19] Yancey, W. E. (2005). Evaluating String Comparator Performance for Record Linkage. *RESEARCH REPORT SERIES (Statistics #2005-05)*. https://www.census.gov/content/dam/Census/library/working-papers/2005/adrm/rrs2005-05.pdf