**Identifying Approximately Duplicate Records**

**in the Thai Language**

Komon Jitsatja1, Intiraporn Mulasastra2, Aphirak Jansang3

Department of Computer Engineering

Faculty of Engineering, Kasetsart University

Bangkok, Thailand

1guitar\_wc@hotmail.com, 2int@ku.ac.th, 3aphirak.j@ku.ac.th

**Abstract**

**The arising need for integrating data from different sources has led to increasing tasks in identifying duplicate records. Most data from different sources are often represented in various standards and may contain some errors; hence we need approximately duplicate data detection to determine similar data representing the same real-world entity. However, the currently available commercial software provides integration modules which do not support detecting duplicate data in the Thai language. This study presents an algorithm for identifying clusters of approximately duplicate records in the Thai language based on the Sorted Neighborhood Algorithm with Duplicate Count Strategy (DCS++) and Innovative Windows algorithm. The Levenshtein Distance Algorithm and Longest Common Subsequence are adapted for approximately matching Thai strings. The results reveal that incorporating Thai string comparison rules in these algorithms can increase f-measures significantly. All of the f-measures in our experiments with Thai data are above 78%, an increase of approximately 10%.**

Keywords: Data duplication; Data cleansing; Duplicate detection algorithm.

**I. Introduction**

In a digital economy era, private and public organizations in many countries, including Thailand have been trying to optimize information system effectiveness by integrating disparate systems. However, data from various sources representing the same real-world entity may differ due to data entry errors or unstandardized formats [[1](#_ENREF_1)]. Related data need to be identified and redundant data need to be removed; hence, approximate duplicate detection algorithms are required, particularly for the Thai language.

Several studies have conducted research on detecting duplicate records in English using standard duplicate elimination algorithms; much progress has already been made toward the identification of duplicate data in English [[2](#_ENREF_2), [3](#_ENREF_3)]. After integrating data from different sources, the duplicate elimination algorithms usually begin with sorting data according to search key attribute values, followed by a pair-wise comparison of consecutive records. String comparison of different languages needs different techniques; however, most of the existing algorithms are for English and other languages –excluding Thai.

The oldest Thai script yet found is from the Sukhothai age, more than 700 years ago [[4](#_ENREF_4)]. There is evidence that some Thai words have been borrowed from Khmer, Pali, and Sanskrit [[4](#_ENREF_4)]. Thai characters consist of 44 consonant letters with 21 phonetic tones, 18 vowels, four tone marks and two diacritic characters [[4](#_ENREF_4)], as shown in Fig. 1.

1. Consonant Letters

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| ก | ข ฃ ค ฅ ฆ | ง | จ | ญ ย | ฉ ช ฌ | ฎ ด |
| ฏ ต | ซ ศ ษ ส | ณ น | บ | ป | ผ พ ภ | ฝ ฟ |
| ม | ฐ ฑ ฒ ถ ท ธ | ร | ล ฬ | ว | ห ฮ | อ |

*Note: The alphabets with the same phonetic tones are presented in the same box.*

|  |
| --- |
| 2. Vowels |
| ะ ั า ำ ิ ี ึ ื ุ ู เ แ โ ใ ไ ฤ ฦ ๅ |

|  |  |
| --- | --- |
| 3. Tone marks | 4. Diacritics |
| ่ ้ ๊ ๋ | ็ ์ |

Fig. 1 Four groups of Thai characters [4, [5](#_ENREF_5)].

Thai is written on four levels; hence, it is more sophisticated in combining characters into words. The main consonants are in a horizontal line, while the vowels can be either left/right or under/above the consonants, depending on the type of vowels. Additionally, tone marks are always above consonants [[4](#_ENREF_4), [5](#_ENREF_5)]. Moreover, Thai words are written consecutively without spaces or punctuation, so the processing of Thai sentences becomes complicated.

The unique characteristic of each phonetic tone is an important criterion in approximate duplicate detection. Thai words are often misspelled by using consonants in the same group. For example, the three consonants (ศ,ษ, and ส), having the same tone, are often been used interchangeably. Moreover, there are eight groups of Thai vowels, classified by their phonetic tone as follows: {อะ}, {อั,ใอ,ไอ}, {อา,อำ}, {อิ,อี}, {อึ,ฤ}, {อื,ฦ,ฤๅ}, {อุ,อู}, {เอ,แอ}, {โอ} [[5](#_ENREF_5)]. Hence, in approximate duplicate detection, we can use the classification of both the consonants and vowels by the phonetic tones to detect similar or misspelled words. Table I shows examples of Thai word errors due to misspelling and typing errors.

TABLE I. Thai word errors

|  |  |  |
| --- | --- | --- |
| Error type | Description | Example |
| Wrong  | * Overtyping
 | พจจนานุกรม🡪พจนานุกรม |
| typing | * Incompletely typing
 | พจานุกรม🡪พจนานุกรม |
|  | * Using wrong consonants
 | พตนานุกรม🡪พจนานุกรม |
|  | * Mis-ordering typing
 | พจนานุรกม🡪พจนานุกรม |
| Missing spelling | * The same phonetic consonants
 | ลพบุรี/ลบบุรี, กรุงเทพมหานคร/กรุงเทบมหานคร/กรุงเทพมหานคอน |
|  | * Analogous consonant tones
 | กระบี่/กะบี่ |
|  | * Using the wrong tone marks
 | ก๋วยเตี๋ยว/ก้วยเตี๋ยว |
|  | * Unclear meaning and uncertain words
 | เนืองนิตย์/เนืองนิจ |

Source: adapted from [Karoonboonyanan, et al. [6](#_ENREF_6)]

Thus, the purpose of this study is to develop an algorithm for detecting approximately duplicate data in the Thai language, based on existing deduplicating algorithms and string matching techniques.

**II. Related work**

In general, a duplicate detection method consists of two main steps. 1) Sorting and comparison: optimal alignment or sorting of input data for analysis. 2) String matching: study of the input data overall and pre-study of the similarity of the dataset. The algorithms applied in our study are explained as follows.

*A. Sorting and Comparison*

1) Sorted Neighborhood Method (SNM): This technique has been developed based on a basic duplicate elimination algorithm to support a broad range of data [[7](#_ENREF_7)]. The basic algorithm compares each record in a dataset with all the other records, using many resources. The SNM purpose is to reduce the number of pair-wise comparisons by fixing the boundaries of targeting data before comparison (fixed size windowing) as shown in Fig 2.



Fig. 2 Sorted Neighborhood Method [[7](#_ENREF_7)]

The important factor of this technique is the sorting key. There is evidence that the results of duplicate detection using SNM are influenced by the selected sort key [[8](#_ENREF_8)]. In addition, multi-pass duplicate detection using SNM with different sort keys, but with the same window size in each pass, can improve detection efficiency, compared to a single-pass SNM. On the other hand, the multi-pass SNM consumes more time in detecting data duplication.

Its efficiency depends on the window size; the higher the window size, the more time consuming, especially for sizes greater than 50. In typical cases, it has been found that increase of 10 window sizes could improve duplicate detection by 5% [[8](#_ENREF_8)]. However, there are some limitations to the SNM as follows [[8](#_ENREF_8)]:

* Data accuracy is sort key dependent.
* Window size is critical for a large data set of duplicate records; if windows with a specified size does not cover all the duplicate records, there would be undetected duplicate records.
* Oversize windows could gain extra-comparing range but consume more time to execute, even though performance is better in terms of detection efficiency.

2) Duplicate Count Strategy (DSC++): This algorithm has been developed based on the SNM with an additional function of adjustable window size, so the unique feature of this technique is the window size varying based on detected duplicate records [[9](#_ENREF_9)].

In executing the DCS++, in the beginning, this algorithm still needs to define an initial window size; then the window size will be increased by adding next w-1 records for each time of duplication detected. In addition to the extendable window size for covering all duplicate records, another benefit is when a duplicate record is detected; this algorithm will enter that record into the skipped list, and then skip comparison of that record with the others. As a result, the number of comparisons will be reduced to w-2 when comparing with SNM. The DCS++ concept is shown in Fig. 3.

 

Fig. 3 DCS++ Concept [[9](#_ENREF_9)]

3) Innovative Windows algorithm (INW): the DCS++ automatically increases the window size of each duplicate detection to cover as many as possible duplicate records. In adopting DCS++, INW adds a window-size shrinking feature. INW defines a terminating count number and will automatically terminate a current window when the count of non-duplicate records detected is greater than the terminating count [[10](#_ENREF_10)]. The INW algorithm has been adapted to improve the performance of DCS++.

*B. String Matching*

1) Levenshtein Distance Algorithm (LD): The purpose of this method is to measure the distance between two strings (s1, s2) [[11](#_ENREF_11), [12](#_ENREF_12)]. The calculated distance is the number of deletions, insertions, or substitutions required to transform s1 into s2. The total time is O(nm), where n and m are the lengths of s1 and s2 respectively.

2) Longest Common Subsequence algorithm (LCS): This algorithm determines the similarity or subsequences in common between two strings (s1, s2) [[13](#_ENREF_13)]. The LCD finds the longest subsequences presented in the two strings. Subsequences may not occupy consecutive positions within the two original strings. The total time consumed by this algorithm is O(nm), where n and m are the lengths of s1 and s2 respectively.

Based on the presented algorithms, this study developed a system for identifying Thai duplicate records.

**III. Our Proposed System for Identifying**

**Thai Duplicate Records**

*A. Developing a Sorting and Comparison Algorithm for the Thai language*

Differences between English and Thai languages [[14](#_ENREF_14), [15](#_ENREF_15)] are such that sorting data in each language requires different string comparison methods. Basically, two strings are compared character by character, from the first position to the last one, regardless of character types (e.g., vowels or consonants). In the Thai language, more details need to be considered. Other than the consonants, Thai vowels, tone marks, and diacritics play important roles in sorting and comparing Thai strings as following [14, 15]:

* The leading vowel appearing before a consonant has lower priority than that following consonant in Thai string sorting.
* Tone marks after a vowel have lower priority than that vowel in Thai string sorting.
* Diacritics in a word are ignored in Thai string sorting unless all the other characters in that word are the same as per the previously mentioned rules.

In addition, when comparing strings, we need to consider that consonants may have the same phonetic tones as other, different but similar characters; they are often used interchangeably. Also, vowels having the same phonetic tones as explained in Part II should be considered as similar characters. In addition, there are many groups of characters that have similar figures (Table II), resulting in misspelling. These phonetic groups of characters are also applied in our string comparison algorithm.

TABLE II. THAI CONSONANTS AND VOWELS GROUPED BY PHONETIC AND HOMOGRAPH

|  |  |
| --- | --- |
| Group Type | Group Result |
| Consonants (in phonetic groups) | {ก}, {ข, ฃ, ค, ฅ, ฆ}, {ง}, {จ}, {ญ, ย}, {ฉ, ช, ฌ}, {ฎ, ด} {ฏ, ต}, {ซ, ศ, ษ, ส}, {ณ, น}, {บ}, {ป}, {ผ, พ, ภ}, {ฝ, ฟ} {ม}, {ฐ, ฑ, ฒ, ถ, ท, ธ}, {ร}, {ล, ฬ}, {ว}, {ห, ฮ}, {อ} |
| Consonants (in homograph groups) | {ก, ถ, ภ, ฎ, ฏ, ฤ, ฦ}, {ข, ฃ, ช, ซ}, {จ, ว}, {ค, ศ, ด}, {ฅ, ต, ฒ}, {บ, ป, ษ}, {ฆ, ฑ}, {ร, ธ, ฐ}, {พ, ฟ, ฬ}, {ผ, ฝ}, {ฉ, น}, {ฌ, ณ, ญ}, {ล, ส}, {อ, ฮ}  |
| Vowels (in phonetic groups) | {อะ},{อั,ใอ,อ}, {อา,อำ}, {อิ,อี}, {อึ,ฤ}, {อื,ฦ,ฤๅ}, {อุ,อู}, {เอ,แอ}, {โอ} |
| Vowels(in homograph groups) | {อะ}, {อั}, {ใอ,ไอ,โอ}, {อา,อำ}, {อิ,อี,อึ,อื}, {ฤ,ฦ,ฤๅ}, {อุ,อู}, {เอ,แอ} |

Source: [Anuntanuruk [16](#_ENREF_16)]

Our input data are collections of customer data having many attributes, namely first name, last name, house number, street name, city, and province. Our system allows users to specify difference weight percentages for each attribute (or columns) according to their importance in identifying string similarity.

*B. Designing System for Identifying Thai Duplicate Records*

The system was designed as shown in Fig.4. After loading data from a specified data source, a user needs to specify sort key attributes (e.g., first name and last name) and their weight percentages for prioritization in our Thai string matching algorithm. Then, a starting window size is required to be specified (e.g., 10, 20, 30, 40, or 50). After that, the user can select a sorting algorithm (DCS++, IW, Multi-pass DCS++, Multi-pass INW). Next, according to the algorithm selected, the system performs sorting following our rules and criteria of Thai string processing.

 

Fig. 4 Our system for identifying Thai duplicate records

The Thai rules that we use in this study have been adapted from the rules shown in Table II, based on commonly misspelled characters. We do not apply all the Thai string processing rules because the more rules applied, the more execution time used. Then, using the LD and LCS algorithms, as well as the Thai rules, the system performs pare-wise comparison in each window. Finally, duplicate data detected will be displayed with duplication scores calculated according to the weight percentages specified for each column in the dataset.

**IV. Experimental results**

Our proposed system adapted existing duplicate detection and string comparison techniques to be suitable for the Thai language. We created a dataset consisting of 568 records of customer data with some duplication for testing our system. The following terms are used to identify the effectiveness of our duplicate detection, including precision, recall, and f-measure. Their meanings are explained as followed.

Precision = # of correct elements identified

 total # of identified elements

Recall = # of correct elements identified total # of relevant elements

F-measure = 2 x Precision x Recall

 Precision + Recall

In this study, we applied deduplication algorithms (i.e., SNM, DCS++, INW, and their multi-pass execution) using Thai string processing rules and compare their f-measure scores and execution times. By using our dataset, we analyzed the performance of all the algorithms by using an initial window size of 10. The threshold for determining string similarity is 92%. The sort key consists of first name and last name. For multi-pass execution, the secondary sort key is the customer address. As shown in Fig. 5, the results reveal a significant improvement of all the algorithms using Thai rules in regard to f-measures.

However, when comparing f-measures among all algorithms using the same configuration, their performances are similar. Testing of the INW with both settings yields the lowest f-measures; this may be due to the dynamic window size in the INW, expanding and shrinking. Hence, some duplicate data might be undetected in the case of shrinking window size.



Fig. 5 Comparison of F-measures in executing algorithms using Thai and English configurations.

We also observed a number of pair-wise comparisons from testing the algorithms with and without Thai settings. The results show that using Thai settings yields only slightly fewer comparisons than using no Thai settings as shown in Fig. 6. However, it is interesting to find that executing INW in both single and multiple passes produced a substantially fewer number of comparisons in both settings.

When executing all the algorithms in multi-passes, the number of comparisons increases significantly (almost double). However, the results in regard to the f-measures were better than single-pass executions (for 6-9%) as shown in Fig. 5.



Fig. 6 The number of pair-wise comparison in executing the algorithms using Thai and English configurations.

By applying Thai string processing rules, the system needs more resources, time and comparisons. As shown in Fig. 7, execution time of the algorithms with Thai settings was a lot higher than those without Thai setting. It should be noted that the INW used the least execution time, which might be because of the number of comparisons in the INW, as explained above (Fig.6), was less than that in the other algorithms.

 

Fig. 7 Execution time comparison of the algorithms using Thai and English configurations.

**V. Conclusion**

The aim of this study was to develop a duplicate record detection system for the Thai language. The deduplication algorithms examined in this study are SNM, DCS++, INW, and their multi-pass execution. For string comparison, this study used LD and LCS algorithms. Since Thai writing script is complex and different from English, we need different techniques for compare Thai strings. This study incorporated Thai string comparison rules suitable for the Thai language into the existing deduplication algorithms to compare their performances.

The results showed that applying Thai rules increased f-measure scores, reflecting better overall performance of the examined algorithms. All of the f-measures in our experiments with Thai data are above 78%, an increase of approximately 10%. On the other hand, the number of comparisons and execution times also increase significantly, when using Thai configurations. However, if the goal is to achieve a higher f-measure (precision and recall), then it is worthwhile to implement a deduplication system using Thai string comparison rules.

As a result of testing the three algorithms, we found that the DCS++ had slightly better performance than the INW. However, the INW was substantially better concerning the number of comparisons and execution time.

It should be noted that selecting an appropriate sort key is important in identifying duplicate records since the sort key is used to group similar records more closely. Consequently, the selected sort key could affect accuracy and execution time. Also, the more Thai string rules that we add into the algorithms, the more number of comparisons and execution time are needed.

This study had a limitation on the number of records in the tested dataset. A larger set of data needs to be used to test for the most suitable set of Thai string rules for deduplication of Thai data. Also, future research should be conducted on testing the algorithms in parallel systems so as to reduce execution time and increase performance.

**Acknowledgment**

The authors would like to acknowledge all referenced researchers for their very helpful work. This research is partially supported by Kasetsart University and advised by the thesis committee in the Department of Computer Engineering, Faculty of Engineering, Kasetsart University.

References

[1] A. Monge and C. Elkan, "An Efficient Domain-Independent Algorithm for Detecting Approximately Duplicate Database Records," in *The SIGMOD 1997 workshop on data mining and knowledge discovery*, 1997.

[2] W. Cohen and J. Richman, "Learning to match and cluster large high-dimensional data sets for data integration," in *The Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, Alberta, Canada, 2002, pp. 475-480.

[3] a. V. E. M. Rehman, "Duplicated Record Detection for Database Cleansing," in *The Second International Conference on Machine Vision*, 2009.

[4] T. Karoonboonyanan, "Standardization and Implementations of Thai language," presented at the Seminar on Enhancement of the International Standardization of Activities in Asia Pacific Region, Japan, 1999.

[5] R. Marukatat, "Matching Entities from Bilingual (Thai/English) Data Sources," presented at the International Conference on Information Communication and Management, 2011.

[6] T. Karoonboonyanan, V. Sornlertlamvanich, and S. Meknavin, "A Thai Soundex System for Spelling Correction," in *The National Language Processing Pacific Rim Symposium*, 1997.

[7] O. H. Akel, "A Comparative Study of Duplicate Record Detection Techniques," Master Degree, Middle East University, Amman, Jordan, 2012.

[8] M. Hernández and S. Stolfo, "Real-world Data is Dirty: Data Cleansing and The Merge/Purge Problem," *Data Mining and Knowledge Discovery,* vol. 2, pp. 9-37, 1998.

[9] U. Draisbach, F. Naumann, and S. S. O. Wonneberg, "Adaptive Windows for Duplicate Detection," in *The 28th International Conference on Data Engineering (ICDE)*, Washington, D.C., USA, 2012.

[10] H. Bano and F. Azam, "Innovative Windows for Duplicate Detection," *International Journal of Software Engineering and Its Applications,* vol. 9, pp. 95-104, 2015.

[11] E. S. Ristad, P. N. Yianilos, and S. Member, "Learning String Edit Distance," *IEEE Transactions on Pattern Analysis and Machine Intelligence,* 1998.

[12] M. Jindarak and S. Nootyaskool, "Implementation of Edit Distance with Thai language," presented at the The Tenth National Conference on Computing and Information Technology (NCCIT2014), Phuket, Thailand, 2014.

[13] L. Bergroth, H. Hakonen, and T. Raita, "A Survey of Longest Common Subsequence Algorithms," in *International Symposium on String Processing and Information Retrieval*, Lisbon, Portugal, 2000.

[14] S. Khamthaidee, "Thai Style Algorithm: Sorting Thai," *Computer Review,* vol. 91, 1992.

[15] T. Karoonboonyanan, "Sorting Thai Words with Punctuation Marks," *NECTEC Journal,* vol. 14, 1997.

[16] T. Anuntanuruk. (2013). Available: <http://tathanapan.blogspot.com/p/blog-page_1037.html>