Matching Entities by Their Thai and English Proper Names

Rangsipan Marukatat, *Member, IACSIT*

***Abstract*—We have developed a framework to match records from different data sources that refer to the same entities, using proper names as the matching keys. There are two challenges to overcome. Firstly, there may be typographical errors. Secondly, some data sources may store the data in Thai characters while some store them in English characters. Thus, Thai-version keys are romanized and compared with English-version keys, using string comparators and rule-based decision function. We report our experimental results and problems encountered, as well as suggest future research directions.**

***Index Terms*—Entity names, record matching, romanization, Thai characters.**

I. Introduction

Many information systems these days process data from many data sources. Therefore, one essential task is identifying records from disparate sources that refer to the same entities. Once matched records are found, they are merged to increase the dimensionality of data, or duplicate ones are removed. This task has several names such as entity matching, record matching, record linkage, or data deduplication.

In testing whether any two records are matched, their keys are compared. Deterministic approach requires them to be exactly equal, whereas approximate approach requires them to be similar to a certain degree. The approximate approach is suitable when the keys are proper names because there may be inconsistencies or typographical errors across different data sources. Firstly, in Thai, an entity's name may be spelled with slight variations such as "พิสณุ", "พิสนุ", "พิศณุ", "พิศนุ", or "พิษณุ". Secondly, some data sources may store the data in Thai characters while some store them in English characters. In such a case, one needs to convert the Thai-version names into English (or English-version names into Thai) so that the keys are written in the same language characters. Although an official romanization standard has been set [1], it is hardly followed in real practice. Hence, "พิสณุ" (or its variant) may be written in English as "Pitsanu", "Pissanu", or "Pisanu".

In short, we must be able to compare an entity's names written in any different variations, either in Thai or English. Our record matching framework is shown in Fig. 1. The rest of this paper is organized as follows. Section II explains Thai writing system. Section III presents record matching method and tool. Section IV reports experimental results, and Section V concludes the paper.

Manuscript received November 18, 2011; revised December 20, 2011.

R. Marukatat is with the Department of Computer Engineering, Faculty of Engineering, Mahidol University, Thailand. (phone: +662-889-2138 ext 6251-2; fax: +662-889-2138 ext 6259; e-mail: egrmr@mahidol.ac.th).

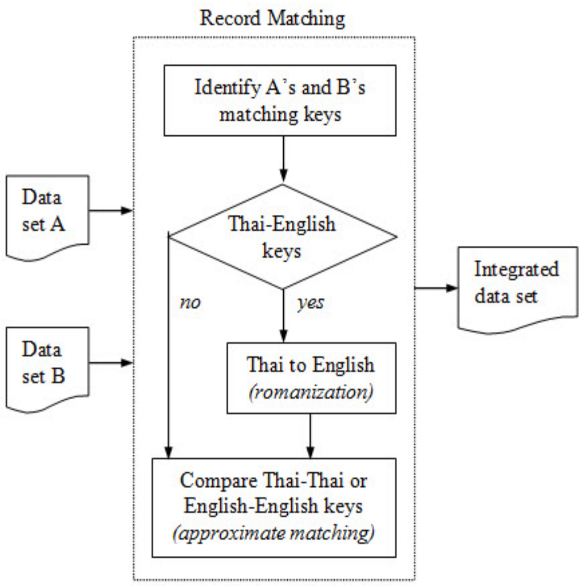

Fig. 1. An overview of record matching task

TABLE I: Thai Characters and Their English Mapping

Thai Consonant	English Lead	English Final	Thai Vowel (□ is consonant)	English
ก	k	k	◌ะ ◌ั ◌า	a
ข ฃ ค ฅ ฆ	kh	k	◌ำ	am
ง	ng	ng	◌ิ ◌ี	i
จ	ch	t	◌ึ ◌ื	ue
ฉ ช ฌ	ch	t	◌ุ ◌ู	u
ซ ศ ษ ส	s	t	เ◌ เ◌ะ	e
ญ ย	y	n	แ◌ แ◌ะ	ae
ฎ ด	d	t	ใ◌ ไ◌	ai
ฏ ต	t	t	โ◌ โ◌ะ เ◌าะ ◌อ	o
ฐ ฑ ฒ ถ ท ธ	th	t	เ◌า	ao
ณ น	n	n	เย เยะ	ia
บ	b	p	เ◌อ เ◌อะ เ	oe
ป	p	p	เอ เอะ	uea
ผ พ ภ	ph	p	◌ัว ◌ัวะ	ua
ฝ ฟ	f	p	ฤ ฤา	lue
ม	m	m	ฦ ฦา	rue
ร	r	n		
ล ฬ	l	n	Consonant Suffix (+ is vowel)	
ว	w	w	+ย	+i
ห ฮ	h	h	+ว	+ua, +ao, +eo
อ	-	-		
Tone Mark (not romanized)			Diacritic (not romanized)	
่ ้ ๊ ๋			็	sound shortener
			์	sound killer

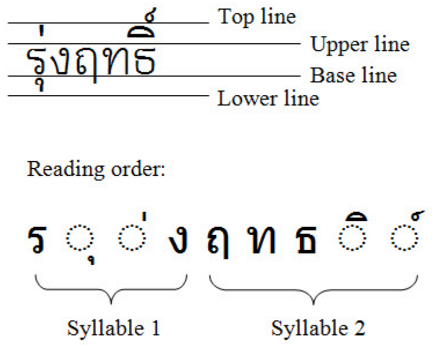

Fig. 2. Thai characters and their reading order

II. Thai Writing System

A. Characters and Syllable Structure

Thai Unicode characters range from \u0E00 to \u0E7F. As displayed in Table I, there are 44 consonants in 21 phonetic groups, 18 vowel symbols (making up single and compound vowels), 4 tone marks, and 2 diacritics [2]. The characters are printed in four lines (Fig. 2). Forward characters are printed on the base line, occupying horizontal space. Dead characters are printed above or below the forward characters. According to Thai encoding standard, TIS620, characters are read from left to right. If there are multiple characters in one column, the reading order starts from forward character, dead character on the lower line, dead character on the upper line, and finally dead character on the top line.

A Thai syllable typically consists of a leading consonant, a vowel, a tone, and a final consonant. Because there is no space between syllables, determining the correct boundary of each one, i.e. syllable segmentation, is a complicated task. Indeed, it is one of challenges in linguistic and information retrieval research, such as [3]-[5].

B. Thai Romanization

Thai proper names are usually derived from Indian origin languages, Pali or Sanskrit. Through a series of conversions from the original language, a name can be written in a few subtly different ways. Misunderstanding or typing errors are therefore very common. To write a name in Roman or English characters, Thai romanization is performed. According to the government proclamation, reported in [6], romanization rules are based on sound transcription. Thai characters are mapped to certain English characters that give the closest sounds (as in Table I). The mapping disregards tone marks, diacritics, and even meaning. Exceptions to these rules were gathered and summarized in [5].

Following the rules, "รุ่ง" and "รุ้ง", pronounced in different tones, are converted into "Rung". The names "พงก์", "พงศ์", and "ภงค์" are all converted into "Phong" as "พ" and "ภ" are members of the same phonetic group while "ก", "ศ", and "ค" are silenced by sound killers.

However, sometimes Pali/Sanskrit romanization is applied to retain the meaning in Pali/Sanskrit, despite inexact sound mapping. For example, an international airport "สุวรรณภูมิ" is officially written "Suvarnabhumi" by Sanskrit romanization, rather than "Suwannaphum" by Thai romanization.

Note that our framework converts Thai-version names into English, not vice versa, because Thai spelling is more diverse than English one (as seen in the aforesaid examples). When a name is romanized, it tends to be close, if not exactly equal, to the English-version name spelled by the person herself. We use Aroonmanakun's romanization ([5], [7]) in this research. It involves three main tasks:

1) Syllable segmentation by predefined syllable patterns. If there are many possible outcomes, the best one is chosen based on trigram probability from a training corpus.
2) Syllable pronunciation. Each syllable is attached with all possible pronunciations. Again, the most probable one is chosen based on probability from another corpus.
3) Romanization. Each syllable's sound is then mapped to English characters using the official standard. A hyphen is added to avoid ambiguity at the syllables' boundaries, so "สอาด" is converted into "Sa-at". A space is added for groups of syllables that form isolable words or subwords, so "ฉัตรสกล" is converted into "Chat Sakon".

III. Record Matching Method and Tool

Suppose that there are two data sources *A* and *B*. Our task is identifying record(s) *b* in *B* that belong to the same entity as record *a* in *A*. The set of matched records can be written $A \times B$ or $\{(a, b) \mid \forall a \in A, \forall b \in B\}$. To determine whether *a* and *b* are matched, *a*'s keys $\{K_{a1}, K_{a2}..., K_{an}\}$ are compared with the corresponding *b*'s keys $\{K_{b1}, K_{b2}..., K_{bn}\}$. This research focuses on approximate matching. The similarity between K_{ai} and K_{bi}, $1 \le i \le n$, is measured, yielding δ_i. Then, a decision function takes $\{\delta_1, \delta_2... \delta_n\}$ and produces a record matching score (Δ) for *a* and *b*.

A. String Comparators

To measure the similarity (δ) between K_a and K_b, both keys are treated as strings. The similarity score is normalized to [0,1] range. String comparators currently used in our research are as follows:

1) *Levenshtein*. This comparator calculates an edit distance based on the minimum number of insertions, deletions, and substitutions of characters to convert one string into another. Each operation incurs a unit cost. The similarity score is a reverse measurement of the edit distance.
2) *Monge-Elkan*. Like Levenshtein, it calculates a similarity score based on edit distance, but assigns decreasing costs to successive operations [8].
3) *Jaro-Winkler*. This comparator counts characters that appear in the corresponding and nearby positions (not farther than half of the length of the shorter string) of both strings, and the conversion from one string into another. Weights are added according to the characters' positions because characters at the tail-end of the string are more likely to differ than those at the beginning of the string [9].
4) *Recursive comparator*. Strings K_a and K_b may consist of tokens or substrings delimited by punctuations, arranged in different orders. Examples are "Harry James Potter"

and "Potter, Harry J.". The recursive method compares every token in one string with every token in the other, using a distance-based comparator (such as Levenshtein, Monge-Elkan, and Jaro-Winkler in this research), and calculates the total similarity score [10].

Our comparators compare Thai and Thai strings or English and English strings. In case that one of them is Thai and the other is English, the Thai string will be romanized prior to the comparison.

B. Decision Function

A decision function combines n comparison results $\{\delta_1, \delta_2 ..., \delta_n\}$ and gives a record matching score. This function can be simple linear regression, expectation-maximization (EM), decision tree, support vector machine, or user-defined rules. Our decision function is rule-based. A user can specify a set of matching rules, as illustrated by Fig. 3 and Fig. 4. Each rule is composed of clauses connected by logical AND operators. For example,

$$\begin{aligned}
\textit{Rule 1:}\quad & JaroWinkler(K_{a1}, K_{b1}) \geq s_{11} \quad AND \\
& JaroWinkler(K_{a2}, K_{b2}) \geq s_{12}. \\
\textit{Rule 2:}\quad & Levenshtein(K_{a1}, K_{b1}) \geq s_{21} \quad AND \\
& Levenshtein(K_{a3}, K_{b3}) \geq s_{22} \quad AND \\
& JaroWinkler(K_{a4}, K_{b4}) \geq s_{23}.
\end{aligned}$$

A similarity threshold s is set for every clause in a rule. A matching score (Δ) according to rule R is the average of all similarity scores (δ's) in that rule.

To find a record in data set B that best matches record a, the following is performed:

```
Candidate_Set = ∅
For each record b in data set B {
    For each rule R {
        If every clause in R is true {
            Calculate matching score Δ
            Add b to Candidate_Set
            Break (i.e. skip remaining rules)
        }
    }
}
Choose b with the highest Δ from Candidate_Set
```

If we want to find multiple matches for a, then candidates can be sorted by their matching scores and the first few can be chosen. On the other hand, if no match is found, then decision rules can be adjusted, e.g. by lowering similarity thresholds or changing string comparators.

C. Data Integration Tool

Our data integration with privacy protection environment, or Dipper, was developed, initially aiming to match records whose sensitive data are concealed [11] (the concealment of sensitive data is not in focus of this paper). It was written in Java, employing Cohen's SecondString API [12] for string comparison. We have been extending it to handle bilingual matching keys, assuming that all characters in a key are either Thai or English.

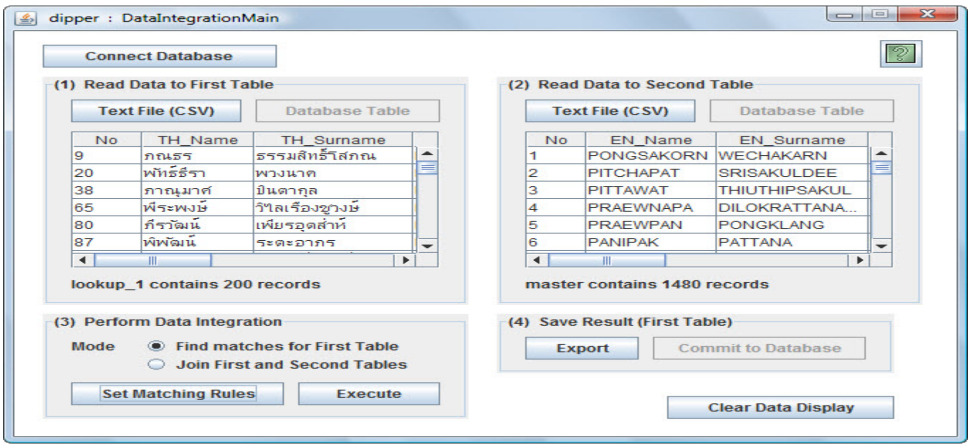

Fig. 3. Data Integrator window

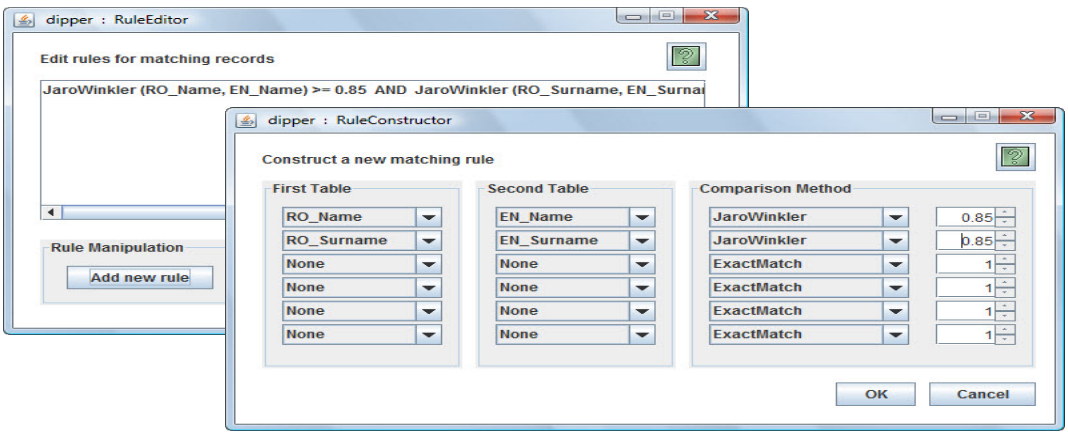

Fig. 4. Rule Editor and Rule Constructor windows

IV. Experiments

A. Experimental Setup

Our Master data set contained 1,480 records of students in the Faculty of Engineering, Mahidol University. Each record had six attributes: ID, name and surname in Thai characters (TH_Name and TH_Surname), name and surname in English characters (EN_Name and EN_Surname), and major of study. We generated five Lookup sets, each containing 200 records randomed from Master. Two experiments were conducted. Their setups are summarized in Table II.

We used the same comparator and similarity threshold for comparing names and surnames. Rule setting was similar to that shown in Fig. 4. But the threshold was only 0.1 in order to allow both correct and incorrect matching, for performance evaluation purpose. Our comparators included Jaro-Winkler (JW), Levenshtein (L), and Monge-Elkan (ME).

In the second experiment, we added typographical errors to 50% of records in both Master and Lookup. The number of induced errors ranged from 1 to 4 per record. Each error was randomed from one of the following common errors:

1) Repetition, e.g. from “Rung” to “Ru**n**ng”
2) Substitution with neighboring character on the keyboard, e.g. from “R**u**ng” to “R**y**ng”
3) Substitution with look-alike character, e.g. from “Run**g**” to “Run**q**”
4) Substitution with shifted/unshifted character, e.g. from “**ร**ุ่ง” to “**ณ**ุ่ง” (shift and “ร” yields “ณ”)

The result of an experimental run was a set of 200 records belonging to the Lookup set used in that run. Each record was expanded with attributes from the matched Master’s and their matching score. We counted the number of *matches* (correct matching), *mismatches* (incorrect matching), and *unmatches* (no matching was found).

B. Results

Fig. 5 shows the average number, across 5 Lookup sets, of correctly matched records in both experiments. Fig. 6 shows the average number of mismatches and unmatches. In the first experiment, all comparators gave more than 90% accuracy. Monge-Elkan was the best one. It was the most optimistic as its matching scores were higher than the others (Fig. 7).

All comparators left nearly identical sets of unmatches in Lookup. It means Dipper could not find any Master’s record that passed the matching rule, despite the similarity threshold being only 0.1. We found that TH_Name/TH_Surname in these records had unconventional spelling that did not fit any syllable pattern in the romanization program. As a result, they were segmented and romanized incorrectly. In many cases, whole syllables were missing from RO_Name/RO_Surname, making them too much different from their corresponding EN_Name/EN_Surname.

With typographical errors in the second experiment, the number of unpronounceable strings increased, leading to even more incorrect romanization and unmatches. On the other hand, adding some errors to TH_Name/TH_Surname did not affect the romanization. For example, “พงก์” and “ภงค์” (with two errors) were both mapped to “Phong”. There was a little drop in Levenshtein’s and Jaro-Winkler’s performance. In contrast, Monge-Elkan was too optimistic and gave too many mismatches. As seen in Fig. 7, its average matching scores for matches and mismatches were both high and close to each other, compared to those of Levenshtein.

TABLE II: Summary of Experimental Setup

	Matching Keys		Compa-rator	Thres-hold
	Master	Lookup		
1	EN_Name	TH_Name *romanized to* RO_Name	JW L ME	0.1 0.1 0.1
	EN_Surname	TH_Surname *romanized to* RO_Surname		
2	EN_Name with typos	TH_Name with typos *romanized to* RO_Name	JW L ME	0.1 0.1 0.1
	EN_Surname with typos	TH_Surname with typos *romanized to* RO_Surname		
Data set summary		Master: 1,480 records Lookup: 200 records × 5 sets		

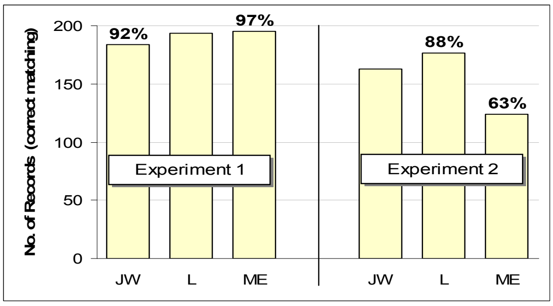

Fig. 5. Average number of *matches* in both experiments

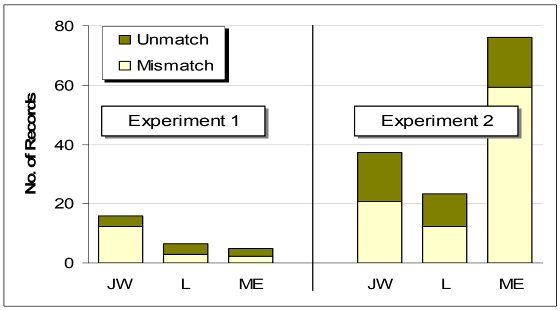

Fig. 6. Average number of *mismatches* and *unmatches* in both experiments

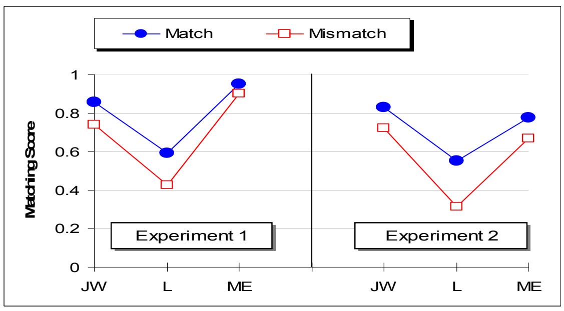

Fig. 7. Average matching scores for *matches* and *mismatches* in both experiment

V. Conclusion

Our research aims to tackle approximate record matching, where matching keys are proper names that may be stored in different variations, either in Thai or English characters. We convert names written in Thai characters into English, and compare them with those written in English characters. The conversion is currently via transcription-based romanization. String comparators namely Levenshtein, Monge-Elkan, and Jaro-Winkler, together with user-defined rules, are employed for comparing names and matching records. Our experiments showed that these comparators were effective. But there were still problems with automatic romanization, especially when typographical errors were present and the boundary of each syllable could not be determined.

Thus, making our data integration program recognize and cleanse some errors beforehand will alleviate the problem. One way is to compare input names with those in a training corpus. To handle unconventionally spelled names, which are increasingly popular, up-to-date samples should be included for training. Besides, we will investigate other approaches to syllable segmentation and romanization. An interesting one, for example, was proposed by Chareonpornsawat and Schultz ([4]). It predicted the syllable's boundary based on entropy measures, and built the pronunciation of an unseen syllable based on the closest one in the training corpus. Another work focusing on romanization was proposed by Tangverapong *et al.* [13].

Lastly, we will incorporate other Thai-English mappings. Dictionary-based techniques can be applied for cases where Thai names are translated rather than romanized, e.g. from "ตึกช้าง" to "Elephant Building", instead of "Tuek Chang". Another mapping is when Thai names are neither romanized nor translated to English. For example, "กรุงเทพมหานคร" is mapped to "Bangkok", not "Krungthep Maha Nakhon" by romanization, or "Great Angel City" by translation.

References

[1] N. Kanchanawan, "Romanization, transliteration, and transcription for the globalization of the Thai language," *The Journal of the Royal Institute of Thailand*, 31(3): 832-841, 2006.

[2] T. Karoonboonyanan, "Standardization and implementations of Thai language," National Electronics and Computer Technology Center (NECTEC), Thailand. Available: http://www.nectec.or.th/it-standards/thaistd.pdf.

[3] C. Wutiwiwatchai and A. Thangthai, "Syllable-based Thai-English machine transliteration," in *Proceedings of the 2010 Named Entities Workshop, ACL 2010*, Upsala, Sweden, 2010, pp. 66-70.

[4] P. Chareonpornsawat and T. Schultz, "Example-based grapheme-to-phoneme conversion for Thai," in *INTERSPEECH 2006 - ICSLP, Ninth International Conference on Spoken Language Processing*, Pittsburgh, PA, USA, 2006, pp. 1268-1271.

[5] W. Aroonmanakun and W. Rivepiboon, "A unified model of Thai romanization and word segmentation," in *Proceedings of the 18th Pacific Asia Conference on Language, Information and Computation (PACLIC)*, Japan, 2004, pp. 205-214.

[6] The Royal Institute of Thailand, "Principles of romanization for Thai script by transcription method," presented at the 8th United Nations Conference on the Standardization of Geographical Names, Berlin, Germany, 2002.

[7] W. Aroonmanakun, Thai romanization [software]. Available: http://www.arts.chula.ac.th/~ling/tts.

[8] A. E. Monge and C. P. Elkan, "The field matching problem: algorithms and applications," in *Proceedings of the 2nd International Conference on Knowledge Discovery and Data Mining (KDD)*, E. Simoudis, *et al.* Eds. AAAI Press, 1996, pp. 267-270.

[9] W. E. Winkler, "The state of record linkage and current research problems," Technical Report RR/1999/04, US Bureau of Census, Washington DC, USA, 1999.

[10] W. W. Cohen, P. Ravikumar, and S. E. Feinberg, "A comparison of string distance metrics for name matching tasks," in *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, Mexico, 2003, pp. 73-78.

[11] R. Marukatat, "Dipper: a data integration with privacy protection environment," in *Proceedings of the International MultiConference of Engineers and Computer Scientists (IMECS)*, Hong Kong, 2009, pp. 750-754.

[12] Second String Project Page. Available: http://secondstring.sourceforge.net.

[13] A. Tangverapong, A. Suchato, and P. Punyabukkana, "Romanization of Thai proper names based on the popularity of usages," in *PAKDD 2009, Lecture Notes in Computer Science, vol. 5476*, T. Theeramunkong, *et al.* Eds. Springer, 2009, pp. 580-587.