Capturing Variants of Transliterated Arabic Names in English Text

Abdusalam F. Ahmad Nwesri
University of Sebha, PO Box 64756,
Gharian, Libya
nwesri@sebhau.edu.ly

Nabila Al-Mabrouk S. Shinbir
Al-tahaddi for computers, PO Box 64756,
Gharian, Libya
n.shinbir@al-tahaddi.com

Abstract

Transliteration is the process of representing words of one language into another using corresponding equivalent phonemes. For example, "Mohammed", "Mohammad", or "Muhammed" are three valid transliterations to the Arabic proper noun "محمد". Transliteration from Arabic to English usually results in several different versions for the same Arabic name causing some names to have more than 40 different versions. Finding transliterated names is a problem in most languages. In English, this problem has been studied by researchers and many techniques have been developed to find transliterated names referring to the same foreign name. Techniques such as string matching and phonetic matching have been used to find similar names. However, some of these techniques were designed to find similar names of English origin and not a specific transliterated names. In this paper we review current techniques used to find variants of the same name and introduce a new technique we specifically developed to find transliterated Arabic names in English text. We developed a data set of more than 25,000 transliterated Arabic names and tested the effectiveness of current and the new technique on finding 115 names within that list. Our results show that our technique is superior to all other techniques. We also present an online system that we developed to find transliterated Arabic names on the web using our technique.

1 Introduction

Transliterated Arabic names in English exhibit many different variants, reaching 40 variants for the same name [Arbabi et al., 1994]. Yet, there is no search engine which deals with finding documents that contain the different versions of the same Arabic name.

Arabic names have become very common in the west due to the increasing immigrants to the western countries. For example, in Britain, the name "Mohammed" — written in 14 different spellings — was the second most popular names for baby boys in year 2006.[i]

Searching for Arabic names in the Internet is of interest to many non-Arabic people due to the fact that many news events are associated with the Middle East. Many news agencies daily report Arabic names written in Latin characters with different spellings used to represent the same person. Non-Arabic speaking users who want to find Arabic names usually type the phonemes of Arabic name or as they hear it in the news. If they mistakenly type one version which is not found in the search engine index, they miss finding relevant documents as the search engine mismatches the user version with versions found in its index. Furthermore, if the search engine succeeds in matching the user version with one of its index terms, it usually matches it with only one version and rarely retrieves documents that contain variants for the same name.

In this paper, we aim at improving finding transliterated Arabic name variants in English text. We also aim at utilizing name matching algorithms to find documents that contain variants of Arabic transliterated names in the user query.

2 Finding Names in English

Most important words in the user query are proper nouns [Abduljaleel and Larkey, 2003]. Proper nouns are names of people, places and organizations. These words usually carry more weight than other words in the query and give an important role to find documents related to the user query. Personal names may have variants within the same language such 'Gale' as 'Gayl' in English. According to Peter [Christen, 2006a], variants of names are generated from different sources such as:

- Confusion between similar letters when reading handwriting such as 'q' and 'g'.
- Entering data manually causes typing the wrong spelling. For example, writing 'Sydeny' instead of 'Sydney'.
- Online entering of data such as over telephone link.
- Limitations in the maximum length of input fields can force people to use abbreviations, initials only, or even disregard some parts of name.

Another major source of errors is transliteration of foreign names. Foreign names are characterized by having different versions [Pirkola et al., 2001, Nwesri et al., 2007]. This is due to the fact that phonemes are transliterated differently from their original language. For example, the sound for the letter "خ" in Arabic as in the word "خالد" has no similar sound in English. This is sometimes represented using the letter "k" as in "Kaled" and sometimes using the letters "kh" as in "Khaled". [Alghamdi, 2005] states that there are about 21 sounds in Arabic that have no similar sounds in English. This creates a problem in writing Arabic names in English.

2.1 Name Matching Algorithms

Finding variants of the same name in English have been investigated by many researchers [Zobel and Dart, 1996, Christen, 2006a]. Several similarity matching algorithms have been developed to find name variants. In the following section we present some of these techniques. Name matching algorithms can be grouped into two main groups: phonetic matching algorithms, and string similarity matching algorithms. Phonetic string algorithms use sounds to match names. One instance of such algorithms is Soundex.

2.1.1 Soundex

The Soundex algorithm, patented in 1918 by Odell and Russell [Hall and Dowling, 1980], is designed to find spelling variations of names. It groups different characters under 6 different codes. It replaces characters by codes except the first characters; disregards vowels, "h", and "w"; and limits the final code to 4 characters. Table 1 shows these groups. The algorithm can be summarized in the following steps:

- Replace all but the first letter of the string by its phonetic code.
- Eliminate any adjacent repetition of codes.
- Eliminate all occurrences of code 0,i.e. eliminate all vowels.
- Return the first four characters of the resulting string.

Using this algorithm, names such as "Nabila", "Nabilah", and "Nabeela" have the same code "N140". However, the name "Nabil" would also have the same code, although they are different. Another algorithm that measures the similarity between two strings is the Levenshtein Edit Distance.

Code	0	1	2	3	4	5	6
Group	a e I o u y h w	b p f v	c g j k q s x z	d t	l	m n	r

Table 1. Groups used by Soundex algorithm to encode names.

		N	a	b	i	l	a
	0 (0,0)	1 (1,0)	2 (2,0)	3 (3,0)	4 (4,0)	5 (5,0)	6 (6,0)
N	1 (0,1)	0 (1,1)	1 (2,1)	2 (3,1)	3 (4,1)	4 (5,1)	5 (6,1)
a	2 (0,2)	1 (1,2)	0 (2,2)	1 (3,2)	2 (4,2)	3 (5,2)	4 (6,2)
b	3 (0,3)	2 (1,3)	1 (2,3)	0 (3,3)	1 (4,3)	2 (5,3)	3 (6,3)
i	4 (0,4)	3 (1,4)	2 (2,4)	1 (3,4)	0 (4,4)	1 (5,4)	2 (6,4)
l	5 (0,5)	4 (1,5)	3 (2,5)	2 (3,5)	1 (4,5)	0 (5,5)	1 (6,5)
a	6 (0,6)	5 (1,6)	4 (2,6)	3 (3,6)	2 (4,6)	1 (5,6)	0 (6,6)
h	7 (0,7)	6 (1,7)	5 (2,7)	4 (3,7)	3 (4,7)	2 (5,7)	1 (6,7)

Table 2. Calculating Edit Distance between the strings "Nabila" and "Nabilah". The final computed distances is in the bottom-right [position (6,7)].

2.1.2 Levenshtein Edit Distance

This algorithm calculates the distance between two strings in terms of the number of insertion, deletions or substitutions required to change one string to another. The edit distance is calculated using the recurrence relation shown in Equation 1. For examples, the edit distance between "nabila" and "nabilah" is 1 (delete "h") and the edit distance between "abdul" and "abdel" is 1 (replace "u" with "e"). Calculating the distance using Edit Distance is shown in Table 2.

$$
\begin{aligned}
&edit(0,\ 0) = 0\\
&edit(i,\ 0) = i\\
&edit(0,\ j) = j\\
&edit(i,\ j) = min[edit(i\text{-}1,\ j) + 1,\ edit(i,\ j\text{-}1) + 1,\ edit(i\text{-}1,\ j\text{-}1) + d(s_i,\ t_i)] \qquad (1)
\end{aligned}
$$

where s is a string with length n indexed using i, t is a string of length m indexed using j, and $d(s[i],\ t[j]) = 0$ if $s[i] = t[j]$, 1 otherwise. The algorithm uses a matrix to align the two strings. It first assigns the values of i and j to the first row and first column respectively. Starting at $edit(1,\ 1)$, and ending at position $edit(n,m)$, the algorithm compares the character of $s[i]$ to the character $t[j]$, and the adjacent previous characters. The final distance is found at $edit(n,\ m)$.

2.1.3 Gram Count

The n-grams of a string are overlapping windows of size n characters. For example, the 3-grams for the string "Nabila" are : "Nab", "abi", "bil" and "ila". and the 3-grams for

"Nabilah" are : "Nab", "abi", "bil", "ila" and "lah". The Gram Count measure uses the number of common n-grams between a string *s* and string *t* to compute their similarity. One measure used by Pfeifer et al. [1996] calculates the similarity between strings *s* and *t* as:

$$GramCount = sim(s,t) = \frac{|G_s \cap G_t|}{|G_s \cup G_t|} \quad (2)$$

where G_s is the set of grams in string *s*, and G_t is the set of grams in string *t*. The absolute sign is used to show that the number of n-grams is returned. For example, if n=3, then the similarity between "Nabila" and "Nabilah" using this measure

$$\text{gramCount}(\text{Nabila}, \text{Nabilah}) = \frac{|\{Nab, abi, bil, ila\} \cap \{Nab, abi, bil, ila, lah\}|}{|\{Nab, abi, bil, ila\} \cup \{Nab, abi, bil, ila, lah\}|}$$

$$= \frac{|\{Nab, abi, bil, ila\}|}{\{Nab, abi, bil, ila, lah\}|} = \frac{4}{5} = 0.8$$

2.1.4 Gram Distance

Another measure uses n-grams is the Gram distance [Ukkonen, 1992], which computes the distance between two strings as:

$$gramDist(s,t) = |G_s| + |G_t| - 2\times|G_s \cap G_t| \quad (3)$$

According to this measure, the distance between "Nabila" and "Nabilah"

$$\text{gramDist}(\text{Nabila}, \text{Nabilah}) = |\{Nab, abi, bil, ila\}| + |\{Nab, abi, bil, ila, lah\}| - 2 \times |\{Nab, abi, bil, ila\} \cap \{Nab, abi, bil, ila, lah\}|$$

$$= 4 + 5 - 2 \times 4 = 1$$

2.1.5 The Dice Measure

Introduced in 1945 [Dice, 1945], the Dice measure computes the similarity between strings *s* and *t* using the equation:

$$Dice(s,t) = \frac{2\times|G_s \cap G_t|}{|G_s| + |G_t|} \quad (4)$$

The similarity between "Nabila" and "Nabilah" when using this measure and 3-grams

$$\text{gramDist}(\text{Nabila}, \text{Nabilah}) = \frac{2 \times |\{Nab, abi, bil, ila\} \cap \{Nab, abi, bil, ila, lah\}|}{|\{Nab, abi, bil, ila\}| + |\{Nab, abi, bil, ila, lah\}|}$$

$$= \frac{2 \times 4}{9} = 0.89$$

3 Related Work

Finding names in English has been studied in depth by many researchers. In this section we report experiments to find name variants in English.

Zobel and Dart [1995] tested the effectiveness of phonetic-matching and string similarity techniques in finding name variants. They used a list of 31,763 distinct English personal names extracted from student names and used 48 randomly-chosen names as queries. Zobel and Dart [1995] used 9 algorithms including Edit Distance, gram- Count, gramDist, and

Soundex. They evaluated the top 200 answers returned by the different algorithms and found that the Edit Distance finds name variants better than other algorithms, with a precision of 63.7%, followed by gramDist and gramCount. The Soundex algorithm perform poorer than other string similarity algorithms. Zobel and Dart [1996] used another list of over than 30,000 distinct English names extracted from the Web to test the performance of phonetic and string similarity algorithms in finding name variants. They randomly selected 100 names as queries from the Melbourne White Pages telephone directory. Their results also show that the Edit Distance and the Q-grams algorithms are better than the Soundex algorithm in finding name variants.

Pfeifer et al. [1995, 1996] used a list of 14,972 distinct names from different sources and randomly chosen 90 names as queries. They tested the effectiveness of finding name variants using phonetic and string similarity algorithms including Soundex, and n-grams. Their results show that all similarity techniques are significantly better than the exact match technique, and that the tailed n-grams perform better than the Soundex algorithm.

Holmes et al. [2004] used a list of 5,819 transliterated Arabic names and 150 queries that have variants in the list to evaluate the performance of their n-grams algorithm. They generate n-grams of Arabic names and then replace certain characters using 45 transformation rules. They compute similarity using the Dice measure presented in the previous section. They achieved an average precision of 90% with a recall of 100%.

Christen [2006a,b] used the test set of Pfeifer et al. [1996] and created three more lists by extracting unique names from healthcare records to evaluate the effectiveness of approximate matching algorithms in finding name variants. They evaluated 24 algorithms and reached the conclusion that name matching algorithms should be chosen based on the data set in hand.

Nwesri et al. [2007] created two data sets to test the effectiveness of finding foreign name variants in Arabic text. The first list contains foreign word variants extracted from Arabic text, while the second contains variants manually transliterated from English words. They clustered name variants and used every variant as a query. They showed that normalizing foreign words by removing vowels and mapping different letter transliterations to one lead to the best results when using the first data set; but that string similarity algorithms such as gramCount and Dice are better than normalization and phonetic algorithms when using the second data set.

Most of the research conducted on finding name variants was conducted to test the effectiveness of string and phonetic matching algorithms in finding name variants along lists of names. There is little research on finding name variants within a text environment. Finding names in this environment is different as many words in the text affect results. For example, the Soundex algorithm considers the name “Nabila”, the word “Nobel”, and the name “Nepal” as similar. We believe that the performance of string similarity and phonetic algorithms would be affected if they were tested to find names within text documents.

Query expansion has been used to expand queries with name variants. For example, Larkey et al. [2003] tested the effectiveness of using several translation and transliteration sources to improving finding Arabic documents using English queries. They translated queries from English to Arabic and generated different variants for proper nouns in English queries. Then they formed Arabic queries from translated words and replaces proper nouns with their generated transliterations. They found that this improves retrieval effectiveness. AbdulJaleel and Larkey [2002] also tested using English queries to search an Arabic text collection. To test the effectiveness of transliteration on retrieval performance, they translated queries using the bilingual dictionary and expanding queries by automatically transliterating names. Their results show that expanding queries using different transliterations generally increases the performance over the baseline.

4 Experiments

In order to test the effectiveness of existing algorithms, we built a dataset of transliterated Arabic names. We started looking for Arabic names written in the Roman script. We used the Google search engine to search for Arabic names using our own transliteration versions of some names. We started with some pages dedicated to Arabic names written in English on the web. We collected a list of 3,245 full names out of which we extracted 2,788 unique single names. We have also crawled lists of graduates from Arabic universities. We crawled the alumni lists from the University of Beirut from year 1964 to 2007.[ii] Our final list had 25,876 unique transliterated Arabic names.

We created the test set by randomly choosing 200 names from our name list. To determine the variants of names for the 200 names in the 25,876 names, we used the *pooling* technique used in IR research to determine relevant documents to a query in a large text collections [Voorhees, 2003]. To collect the possible variants for each name in our data set, we used the Dice, Edit Distance, Gram count, Gram Distance, and Soundex algorithms (see Section 2.1) to find the possible variants for that name. As all algorithms except the Soundex, return a ranked list of variants; we add the first 10 variants returned by each algorithm to the pool. For every name, we removed repeated variants and left only the unique variants. Then we manually refined that list and left the actual variants. If the name has no variants, we remove it from the test set. Our final test set has 115 names with their variants.

5 Evaluation of Existing Algorithms

We run the different algorithms using the test set. For each algorithm, we calculate the average precision and recall. For every name, we use the individual algorithms to search our data set and find its variants. We calculate recall and precision as follows:

$$Recall = \frac{The\ number\ of\ correct\ variants\ returned}{The\ number\ of\ all\ correct\ variants}$$

$$Precision = \frac{The\ number\ of\ correct\ variants\ returned}{The\ number\ of\ all\ returned\ variants}$$

As an example, suppose that the name "Nabila" has three variants (Nabila, Nabeela, and Nabilah), and an algorithm "X" returns (Nabila, Nabeela, Nabil, Nabeel, Nabel) as variants of Nabila. Then, the precision of algorithm X is $\frac{2}{5} = 0.4$, the recall is $\frac{2}{3} = 0.6$.

In many cases algorithms show better recall than other algorithms but have lower precision or vice versa. Another measure that combines precision and recall is called *F* measure [Baeza-Yates and Ribeiro-Neto, 1999]. A harmonic version of this measure is called *F*1 and is computed as:

$$F_1\ measure = \frac{2 \times Precision \times Recall}{Preciaion + Recall}$$

Accordingly the *F*1 measure is $\frac{2\times0.4\times0.6}{0.4+0.6} = 0.48$.

Algorithm	Relevant Retrieved	Retrieved	Recall	Precision	*F*1 Measure
Exact Match	115	115	0.236	**1.000**	0.382
Soundex	398	5,654	**0.816**	0.070	0.129
Dice	175	340	0.359	0.515	0.423
Edit Distance	279	590	0.572	0.473	**0.518**
Gram Count	153	250	0.314	0.612	0.415
Gram Distance	130	194	0.266	0.670	0.381

Table 3. Best results obtained by different algorithms. The number of actual relevant name variants is 488.

We compute the final precision and recall for each algorithm by averaging the result of the 115 names in our list. As our objective is to get the best possible variants to a name, we calculate the precision over 100% recall of each algorithm. For ranked algorithms, we determined the best threshold at which the algorithm achieves the best precision and recall by running algorithms at the different threshold values and observing results. Table 3 shows the best results produced by the different algorithms compared with the exact match technique.

Results show that the exact match technique, where we compare words without any processing, captures 24% of the names. This shows the importance of using techniques to find other variants (around 75%).

The Soundex algorithm captures around 82% of variants, however, the precision is very low. The Edit Distance algorithm captures about 57% of the variants — twice as many as the exact match — however, the precision is only 47%. The gramDist algorithm has the best precision (67%), with only 15 more variants captured than the exact match. The gramCount algorithm has a slight lower precision than the gromDist, but it finds more variants (Recall is 31%). The Dice algorithm has a precision of 51% and a recall of 37%.

As can be seen from the results, none of the algorithms captures all the variants. All algorithms either have a low recall or a low precision. Applications such as search engines, that require direct interaction with users require higher precision, as the users usually look for the first few pages returned. Using an algorithm with lower precision to expand the query causes many unrelated words to be used in the user query. This affects the result of search engine as non-relevant documents with unrelated names would be also retrieved and might be ranked at the first top 10 retrieved documents. We need to consider precision and increase recall to capture as much variants as possible.

In the following section, we describe the algorithm that we implemented specifically to find Arabic name variants. We then compare its effectiveness with other algorithms.

6 The EngNORM Algorithm

According to results in the previous section, we decided to implement another algorithm that finds Arabic name variants. We created our algorithm based on our knowledge of Arabic names in English and the way they are written and based on our observation on names collected from the web. In order to introduce a new algorithm that finds Arabic name variants, we considered the mappings between Arabic and English characters presented in Abduljaleel and Larkey [2003], and analyzed a sample of transliterated Arabic names. We observed the following differences in written Arabic names:

- Names that start with the definite article "الـ" are transliterated differently to either "Al", "Al-", "El", or "'El-".
- Some Arabic characters such as "غ" and "خ" are mapped to more than one English character usually by adding the letter "h" after their respective translated equivalent.

Algorithm	Relevant Retrieved	Retrieved	Recall	Precision	*F*1 Measure
ExactMatch	115	115	0.236	**1.000**	0.382
Soundex	398	5,654	**0.816**	0.070	0.129
Dice	175	340	0.359	0.515	0.423
EngNORM	227	274	0.465	0.828	**0.596**
Edit Distance	279	590	0.572	0.473	0.518
Gram Count	153	250	0.314	0.612	0.415
Gram Distance	130	194	0.266	0.670	0.381

Table 4. EngNORM results compared to other algorithms. The number of actual relevant name variants is 488.

- Compound names that start with "عبد" are transliterated differently to either "Abdul", "Abdul-", "Abdel", "Abdel-", "Abdu", or "Abde". For example, the name "عبدالسلام" is transliterated to "Abdussalam", "Abdul-salam", Abdel-salam", or "Abdulsalam".
- The Arabic feminine singular ending "ـة" is transliterated to either "a" or "ah". For example, "جميلة" is transliterated to "jamila" or "jamilah".
- Single Arabic characters are in some cases mapped to one English characters and in some others mapped to double English characters. For Example, the character "ـمـ" in the name "محمد" is sometimes transliterated to a single "m" as in "mohamed" and sometimes transliterated to a double "mm" as in "mohammed".

Based on these observations, we implemented our new algorithm which normalizes these errors and maps different English characters used to represent the same Arabic character to one characters. The new algorithm is called EngNORM and works as follows:

- Convert all letters to lower case.
- Remove double character.
- Remove "h" if proceeded by "g", "d", "t", or "k".
- Replace "u" with "e" in the sequence "dul".
- Remove the hyphen character "-" and final "h".
- Replace "q" with "k", "g" with "j", "p" with "b", "v" with "f", "o" with "u", and "e" and "y" with "I".
- Remove a vowel only if it is proceeded by another vowel and followed by a consonant.

Results of running our algorithm on the same test set are shown in Table 4. Results show that our algorithm is better than all other algorithms, as it produces better precision while doubling the recall value. Our algorithm captures as twice variants as the exact match algorithm (47%) at a precision of 83%. It performs significantly better than the Edit Distance algorithm (*t*-test, $p<0.05$), and significantly better than other algorithms in the confidence level of 99% ($p<0.001$).

7 Finding Variants on The Web

In order to find variants of Arabic names, we implemented a system that utilizes our algorithm to find variants of the input name and expands the user query with its possible variants. To differentiate between English native words and Arabic names, we used an English dictionary

`

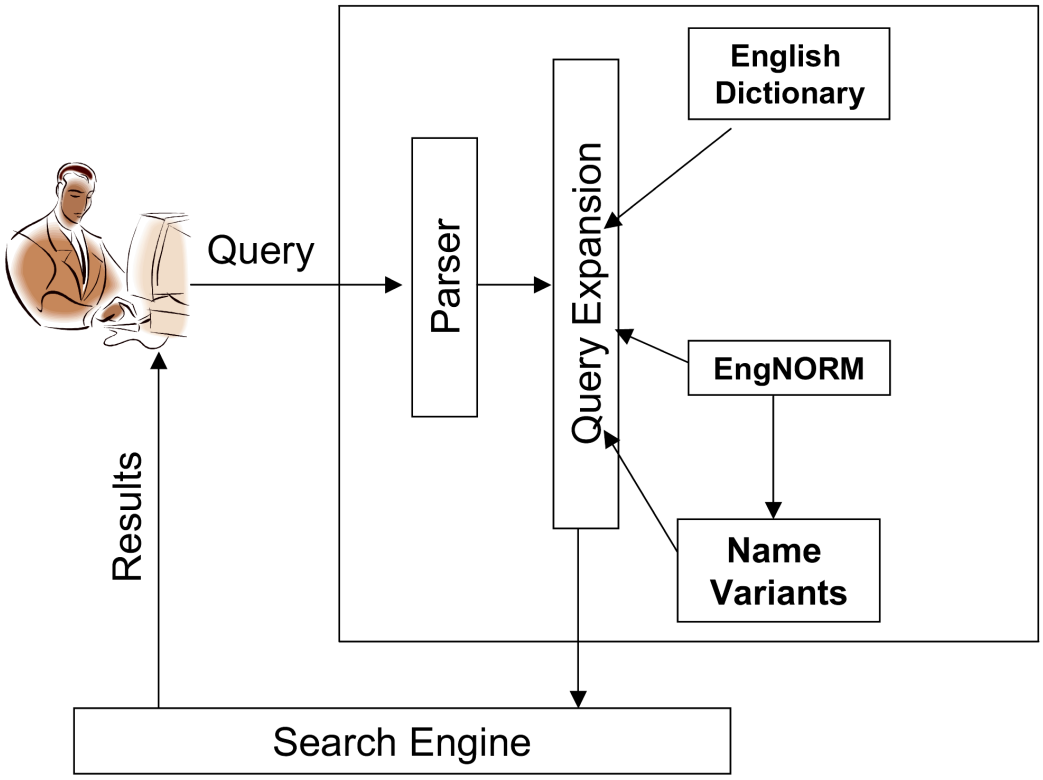

Figure 1. System architecture. The system has two main components: parser and expander.

to avoid expanding English native words. We also used the EngNORM algorithm to create a list of Arabic name variants form our 25,876 crawled Arabic names. The list contains name variants indexed by their normalized version. For example “nabila”, “nabeela”, “nabilah” are indexed by “nabila”.

Figure 1 shows the design of our system. The system begins by parsing the user query and determining non-English words by checking then against the English dictionary. It then checks the normalized word against our index in the name variants list. If the word is not found in the English dictionary, and it is found in the name list, we fetch its variants and add them to the original query. If it is not found in our list, we add it as a new entry to the variants list. This allows our list to grow and include new name variants. We send the final expanded query to the search engine. The user receives the results. Figure 2 shows the results of searching Google using the Arabic name “Nabila”, while Figure 3 shows the Google results using our system. It is clear that the top retrieved pages are not the same and our system considers other name variants. The system can be tested on http://www.al-tahaddi.com/ANS/.

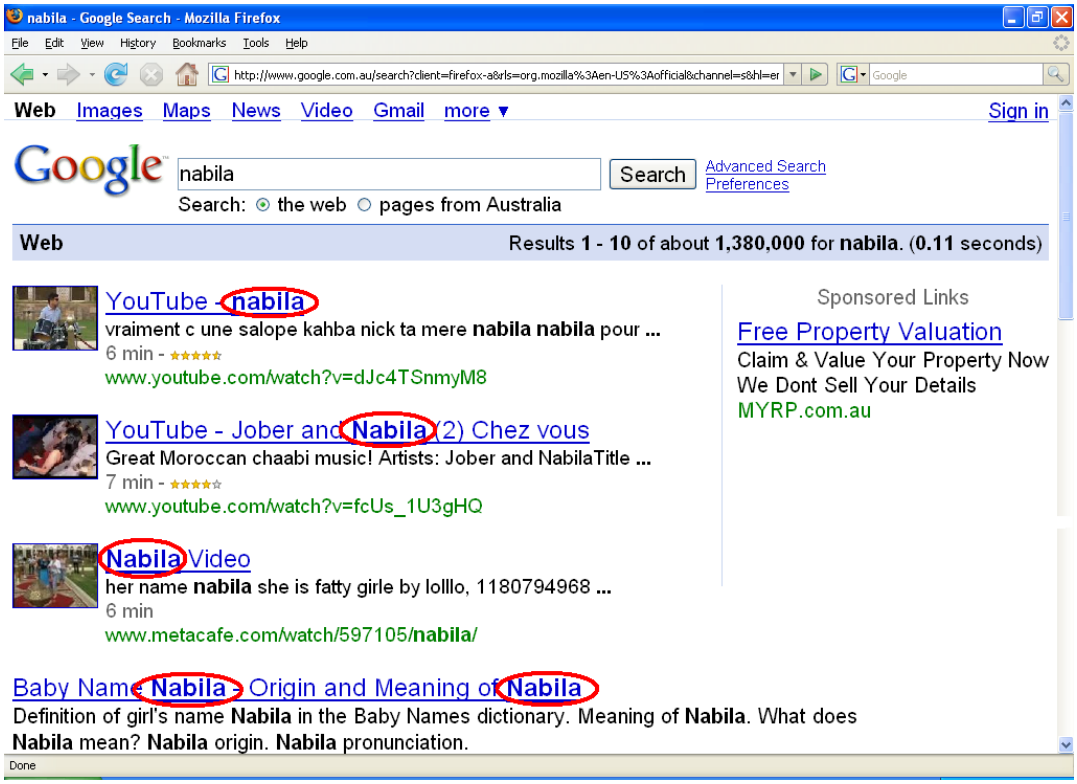

Figure 2. Search using the Google search engine with the transliterated Arabic name "Nabila". Only documents containing "Nabila" are retrieved.

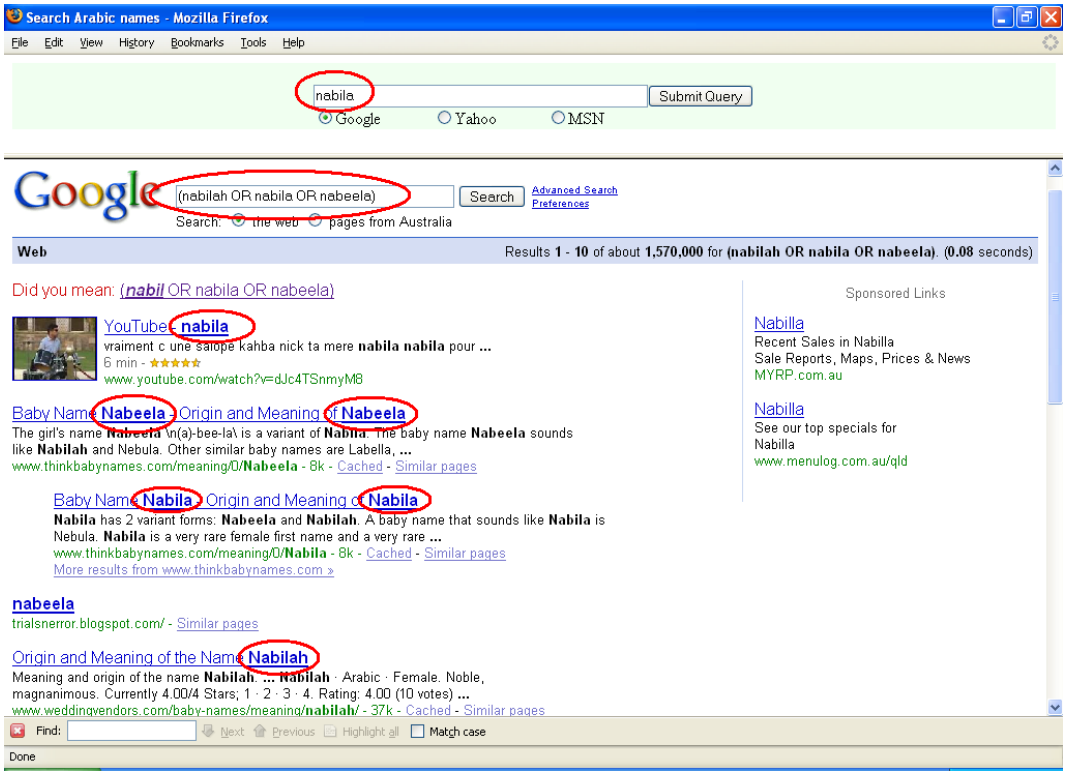

Figure 3. Search using the Google search engine with the expanded query generated by our system for the transliterated Arabic name "Nabila". Documents containing "Nabila", "Nabeela", and "Nabilah" are retrieved.

8 Conclusions

We have implemented an algorithm that finds variants of transliterated Arabic Names. We compared this algorithm with other algorithms used to find name variants in English. We have shown that our algorithm outperformed other algorithms and have implemented a system that utilizes this algorithm to expand the user query with name variants in order to find documents contain such variants using major search engines.

9 References

AbdulJaleel, N. and Larkey. L., English to Arabic transliteration for information retrieval: A statistical approach. Technical Report IR-261, University of Massachusetts, 2002.

AbdulJaleel, N. and Larkey. L., Statistical transliteration for English-Arabic cross-language information retrieval. In *Proceedings of the International Conference on Information and Knowledge Management*, pages 139–146, New Orleans, LA, USA, 2003. ACM Press.

Alghamdi, M., Algorithms for romanizing Arabic names. *Journal of King Saud University: Computer Sciences and Information.*, 17:1–27, 2005. In Arabic.

Arbabi, M., Fischthal, S. M., Cheng, V. C., and Bart, E., Algorithms for Arabic name transliteration. *IBM Journal of Research and Development*, 38(2):183–194, 1994.

Baeza-Yates, R. A. and Ribeiro-Neto, B. A., *Modern Information Retrieval*. ACM Press / Addison-Wesley, 1999. ISBN 0-201-39829-X. URL sunsite.dcc. uchile.cl/irbook/.

Christen, P., A comparison of personal name matching: Techniques and practical issues. Technical report, The Australian National University, September 2006a.

Christen, P., A comparison of personal name matching: Techniques and practical issues. *Sixth IEEE International Conference on Data Mining - Workshops (ICDMW'06)*, 0:290–294, 2006b.

Dice, L. R., Measures of the amount of ecologic association between species. *Ecology*, 26(3):297–302, July 1945. P. A. V. Hall and G. R. Dowling. Approximate string matching. *ACM Computing Surveys*, 12(4):381–402, 1980.

D. Holmes, Kashfi, S., and Aqeel, S. U., Transliterated Arabic name search. In *The 3rd IASTED International Conference on Communications, Internet, and Information Technology*, pages 267–273, St. Thomas, US Virgin Islands, 2004.

Larkey, L., AbdulJaleel, N. and Connell, M., What is a name?: Proper names in Arabic cross language information retrieval. Technical Report IR-278, University of Massachusetts, 2003.

Nwesri, A. F., Tahaghoghi, S., and Scholer, F., Finding variants of Out-of-Vocabulary Words in Arabic. In *Proceedings of the 2007 Workshop on Computational Approaches to Semitic Languages: Common Issues and Resources*, pages 49–56, Prague, Czech Republic, June 2007. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/W/ W07/W07-0807.

Pfeifer, U., Poersch, T., and Fuhr, N., Searching proper names in databases. In R. Kuhlen and M. Rittberger, editors, *Hypertext - Information Retrieval - Multimedia, Synergieeffekte elektronischer Informationssysteme, HIM '95*, volume 20 of *Schriften zur Informationswissenschaft*, pages 259–275, Konstanz, April 1995. Universit¨atsverlag Konstanz.

Pfeifer, U., Poersch, T., and Fuhr, N., Retrieval effectiveness of proper name search methods. *Information Processing & Management*, 32(6):667–679, 1996.

Pirkola , A., Hedlund, T., Keskustalo, H., and Järvelin, K., Dictionary-based cross-language information retrieval: Problems, methods, and research findings. *Journal of Information Retrieval*, 4(3-4):209–230, 2001.

Ukkonen, E., Approximate string-matching with q-grams and maximal matches. *Theory Computer Science.*, 92(1):191– 211, 1992.

Voorhees, E. M., Overview of the TREC 2003 question answering track. In *Proceedings of the Text Retrieval Conference (TREC)*, pages 54–68, 2003.

Zobel, J. and Dart, P., Finding approximate matches in large lexicons. *Software - Practice and Experience*, 25(3):331– 345, 1995.

Zobel, J. and Dart, P., Phonetic string matching: lessons from information retrieval. In *The Nineteenth annual international ACM SIGIR conference on Research and development in information retrieval*, pages 166–172, New York, NY, USA, 1996. ACM Press.

[i] http://www.timesonline.co.uk/tol/news/uk/article1890354.ece

[ii] http://www.bau.edu.lb/Alumni/english_branch.html